175 PTAS

85

MI COMPUTER

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

RED ALERT

V:42 K:00 E:9350 C:0
θ:+18 Φ:-12 R:+053

Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VIII-Fascículo 85

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 118509
Impreso en España-Printed in Spain-Septiembre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.



Dominar la materia

Alan Adler

Juegos mentales
El concepto de *mindlink* o conexión mental suprime la etapa mecánica intermedia de traducir los impulsos del jugador en señales comprensibles para el ordenador. Al permitir una relación más directa entre usuario y máquina, la *mindlink* ofrece un software más amable y una entrada más rápida

Controlar un ordenador por medio de la mente ya no es sólo una idea fantástica creada por la ciencia-ficción, sino una palpable realidad

Desde la época de los primeros ordenadores el teclado ha sido el principal dispositivo de entrada, el medio para introducir información en el ordenador. Probablemente sea el vehículo más eficaz para entrar texto y, en menor grado, cifras, al menos hasta que se desarrollen sistemas de reconocimiento de voz para fines generales. Pero, por muy adecuado que sea el teclado para entrar texto, no es necesariamente el mejor dispositivo para entrar otros tipos de información. Si usted desea entrar rápidamente datos de direcciones, como en muchos juegos, es preferible una palanca de mando o un mando de bola y, por lo general, el ratón, el lápiz óptico o la pantalla sensible al tacto representan los mejores medios para entrar datos posicionales, como cuando se realizan selecciones de menús. Un lector de códigos de barras es un método conveniente para entrar largos números en código o seriados.

Todos estos dispositivos de entrada, sin embargo, adolecen de similar inconveniente: son indirectos. Aunque una palanca de mando es más adecuada que un teclado para practicar juegos (p. ej., para desplazar la nave espacial hacia arriba, uno empuja hacia adelante el bastón de la palanca de mando), aún representa una interrupción, un paso entre lo que uno desea que suceda en el ordenador y lo que realmente sucede. Uno piensa "hacia arriba", traduce mentalmente esta idea en "palanca de mando hacia adelante" y luego, físicamente, la empuja. Para una interacción más rápida y más directa entre el usuario y el ordenador necesitamos eliminar este paso intermedio: ¿por qué no sólo pensar "hacia arriba" y lograr que el ordenador responda directamente al pensamiento?

Imagínese lo que sería jugar al *Defender* simplemente pensando "arriba", "abajo", "girar", "disparar", etc., o escribir una carta pensando sólo en las palabras. El procesador de textos telepático está aún a algunos años de distancia, pero el juego *Defender* controlado mediante el pensamiento es en la actualidad una realidad, hecha posible en virtud de un concepto que se conoce como *mindlink* o conexión mental.

En el concepto de *conexión mental*, el pensamiento (o, con mayor precisión, los cambios fisioló-

Notas mentales

En la fotografía vemos una demostración de un prototipo del sistema GSR operando con un ordenador Apple. Se han desarrollado sistemas similares para ser utilizados con los PC IBM y Commodore, pero es probable que el desarrollo de software eficaz lleve aún algún tiempo. Roger Dilts, presidente de Behavioral Engineering y autor de software GSR, está interesado en combinar *mindlink* con NLP (*neuro-linguistic programming*: programación neurolingüística), rama de la psicología que ha estado en gran parte dedicada al estudio del aprendizaje. Mediante el empleo de la técnica GSR para controlar el estado mental del usuario, el software puede ir comprobando el grado de tensión de éste e ir regulando el flujo del programa. Roger Dilts piensa que los programas educativos, por ejemplo, podrán evaluar la respuesta emocional del estudiante ante el material presentado. Si la presentación es demasiado compleja, el ordenador captará la tensión emocional y permitirá cambiar el ritmo del programa o simplificar el tema de estudio



gicos que se producen a consecuencia de los cambios en los patrones del pensamiento) se utiliza para controlar un dispositivo electrónico. Para el ordenador este dispositivo es como una interface, de forma muy similar a una palanca de mando o cualquier otro dispositivo de entrada. La conexión mental se basa en lo que se conoce como "fenómeno GSR", en función del cual los cambios en el estado emocional del individuo se manifiestan en la conductividad eléctrica de la piel. El usuario se conecta mediante electrodos a un medidor de resistencia. Las señales de este medidor se envían a la puerta para el usuario de un ordenador y son interpretadas y obedecidas mediante un software escrito especialmente. La empresa Behavioral Engineering, con sede en California, ha basado en esta técnica tanto juegos como software práctico, y entre sus productos en este campo hay una versión simplificada del *Defender*. Se sabe, asimismo, que otras importantes empresas de ordenadores han estado trabajando en este campo durante los últimos años, en especial Atari, pero sólo recientemente han obtenido cierto nivel de éxito.

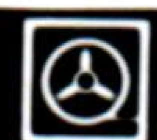
En el *Defender* convencional el jugador controla una nave espacial en órbita alrededor de un planeta. El juego consiste en disparar contra las naves de los alienígenas sin que éstas le disparen a uno y sin estrellarse contra el planeta. En la versión de Behavioral Engineering, uno sólo controla la altura de la nave: ¡pero la gran diferencia es que uno la controla a través del pensamiento! La empresa ha diseñado una interface GSR para el Apple IIe. El usuario simplemente coloca los dedos índice y corazón de una mano sobre un dispositivo similar a un ratón, que mide la resistencia a través de los dos dedos y envía los valores resultantes al ordenador. El soft-

Principios del GSR

El concepto de conexión mental se basa en un fenómeno que se conoce bajo tres denominaciones alternativas: GSR (*galvanic skin response*: respuesta galvánica de la piel), PGR (*psychogalvanic reflex*: reflejo psicogalvánico) y EDR (*electrodermal reflex*: reflejo electrodérmico). En este capítulo hemos utilizado el término GSR. Éste alude a cambios en la conductividad eléctrica de la piel que corresponden a cambios en el estado emocional del individuo. La experimentación ha demostrado que, cuanto más tensa está una persona, menor es la resistencia eléctrica de su piel. La aplicación más conocida del GSR son los *polígrafos* o *detectores de mentiras*. Aunque el fenómeno GSR se ha estudiado desde el siglo XIX, se sabe muy poco sobre sus causas. La teoría original afirmaba que el sudor producido por la excitación o la ansiedad actuaba como un conductor electrolítico, disminuyendo, por tanto, la resistencia de la piel. Sin embargo, experimentos más recientes han arrojado dudas sobre esta simplista teoría. No obstante, se sabe que el GSR está directamente relacionado con el grado de tensión existente en el sistema nervioso simpático. Éste, a su vez, depende del sistema nervioso central y, por consiguiente, del cerebro: de allí la posibilidad de controlar un ordenador mediante el pensamiento. Los cambios en la actividad del cerebro producen cambios en el estado del sistema nervioso central; éste produce un cambio en el estado del sistema nervioso simpático, que conduce a cambios en la resistencia de la piel. Y la variación de una corriente eléctrica es la base de toda forma de dispositivo de entrada



Ian Dobbie



ware está diseñado de modo tal que un aumento en la corriente (una resistencia baja originada por una tensión aumentada) produce la elevación de la nave, mientras que una disminución de la resistencia hace que la nave descienda. La idea, por supuesto, no es otra que la de controlar estos movimientos para alinear la nave con los objetivos que se acercan.

El término *conexión mental*, que parece haber sido acuñado por Atari, quizá sea un tanto impreciso, porque la conexión se establece con el sistema nervioso y no con la mente. Ésta es, no obstante, un área oscura: si los dos son interdependientes, ¿tiene sentido separarlos? La mayoría de las personas encuentran que, después de 20 minutos de juego, pueden ejercer un grado de control bastante alto, a menudo sin ser conscientes de la forma en que lo hacen. Algunas personas tensan y relajan su cuerpo ligeramente de forma consciente, mientras que otras simplemente piensan "arriba" o "abajo" y dejan que su sistema nervioso haga el resto.

Además de los juegos, la conexión mental posee aplicaciones más prácticas. Las personas que se hallan totalmente paralizadas conservan aún la capacidad para generar efectos GSR conscientemente, a pesar del hecho de no tener control sobre sus músculos. Ya se ha conectado con éxito un dispositivo GSR al robot Topo a través de un ordenador Apple, permitiendo que una persona paralítica controle el robot. Otra aplicación interesante se realiza en el espacio, en condiciones de ausencia de gravedad. Sin gravedad para equilibrar la fuerza ejercida, puede ser sumamente difícil operar controles mecánicos. Un GSR podría ser un sustituto ideal para muchos controles mecánicos.

Entre las aplicaciones aún más insólitas se incluye el software perceptor del estado de ánimo. El software educativo, en especial, se podría beneficiar con la realimentación GSR del estado interior del usuario. El individuo podría llevar una muñequera que contuviera los electrodos y dejar que el software calibrara su estado normal relajado. Luego, si en cualquier punto del programa el dispositivo GSR registrara un notable aumento de tensión, se podría suponer que el usuario tenía dificultades y actuar en consecuencia. ¡Hasta el software de gestión podría detectar estrés en el usuario y sugerirle una pausa para tomarse un café!

La rapidez a la que se desarrolle la tecnología GSR dependerá en parte de la introducción de dispositivos de mayor precisión y que respondan con más rapidez, y, en parte también, a nuestra capacidad para perfeccionar nuestra interpretación de los efectos. En cuanto a lo primero, el problema es que, típicamente, la respuesta GSR se produce dos segundos después del acontecimiento y tarda entre dos y diez segundos en desaparecer. Los dispositivos actuales superan hasta cierto punto este inconveniente al medir la velocidad del cambio de la respuesta en lugar de la intensidad de ésta; no obstante, es preciso aún ir más allá en este sentido. El segundo problema radica en la pobreza de las deducciones que somos capaces de realizar a partir de la realimentación GSR. Sabemos que una disminución rápida de la resistencia de la piel indica algún tipo de estrés, pero todavía no somos capaces de determinar si la tensión es agradable o dolorosa.

Sin embargo, a pesar de los problemas, el GSR ofrece algunas posibilidades apasionantes.



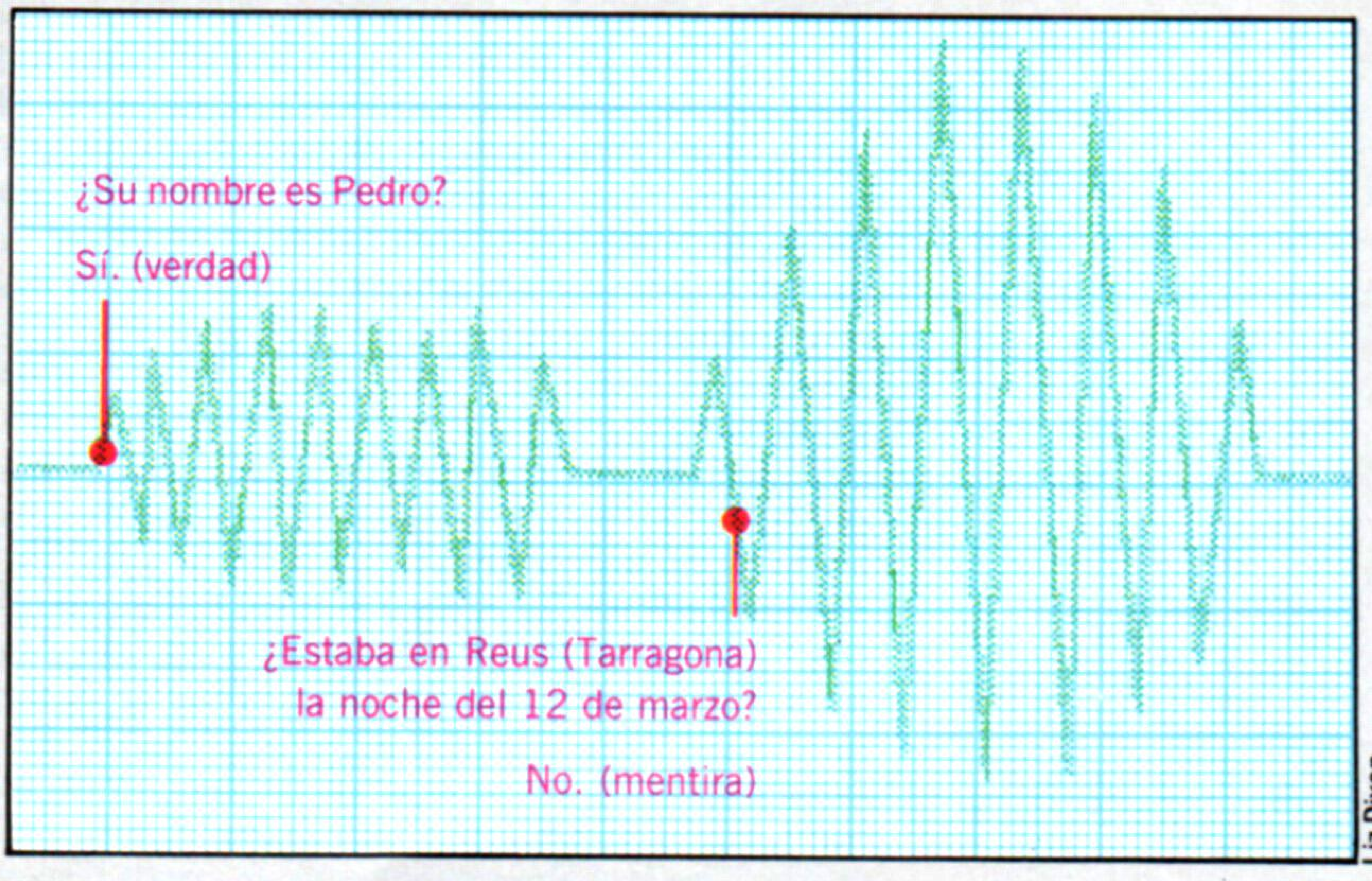
Associated Press

El detector de mentiras

Uno de los principales problemas con que se encuentra la policía y la justicia al realizar una investigación y posterior juicio reside en saber si un sospechoso o un testigo está mintiendo. Si bien un observador experimentado puede detectar mínimos indicios (cambios en la coloración de la piel y la respiración, p. ej.), la medición de estos factores no se ha definido ni catalogado, ni tampoco son admisibles como evidencia. Por este motivo, se ha investigado profundamente para desarrollar un medio objetivo de apreciar la diferencia entre afirmaciones verdaderas y falsas. Uno de los resultados de tal investigación ha sido el *polígrafo* o *detector de mentiras*. En realidad éste no es más que un medidor de resistencia. Se basa en la teoría de que la tensión de un individuo aumenta significativamente al decir una mentira u oír palabras clave relacionadas con el delito, y que este aumento de tensión se refleja en una disminución de la resistencia de la piel. El polígrafo es objeto de gran controversia. Tenido en gran estima por agencias de investigación, en particular en Estados Unidos, sus detractores arguyen que no se sabe lo suficiente acerca del GSR como para interpretarlo de forma fiable, y que diferentes personas pueden reaccionar de forma muy distinta, independientemente de su culpabilidad o inocencia

Para obtener un empleo
En ocasiones se utiliza el GSR para evaluar a los candidatos a un empleo: un detector de mentiras ayuda a calificar las respuestas de un individuo a las preguntas que se le formulan. En la fotografía vemos a un agente de ventas a quien se le están mostrando los resultados de un test que acaba de realizar

Mentiras diáfanas
Un detector de mentiras mide los cambios en el GSR provocados por la respuesta emocional del individuo ante la pregunta formulada. Sin embargo, en la vida real es muy poco probable que un detector de mentiras ofrezca resultados tan evidentes como el que vemos aquí. Las reacciones emocionales varían a tenor de la sensibilidad del individuo ante ciertos temas (p. ej., si el tema le resulta embarazoso) y no dependen sólo de la veracidad o falsedad de las respuestas



Liz Dixon

Estructuras dinámicas

El PASCAL concede una importancia primordial a la precisión de las técnicas de programación al tratar archivos

Ahora podemos seguir desarrollando nuestra base de datos pensando en algunas variables esenciales y en un algoritmo informal.

```
VAR
  lista   : ListaRegistro;
  tamaño : Cardinal;
BEGIN
  tamaño :=0; {tamaño activo de la lista}
  {leer los datos en la lista, actualizando tamaño}
  {clasificar elementos de tamaño en el orden
  correcto}
  {imprimir elementos de tamaño de la lista}
END.
```

Para traducir este algoritmo informal en algo más tangible necesitamos expresar las tres etapas principales del proceso como llamadas a procedimientos (con nombres adecuados) y listar todos los datos conocidos que requerirá cada uno en forma de una lista de parámetros.

Una vez realizado este sencillo paso, podremos escribir todos los bloques de procedimiento como un esqueleto. Por ejemplo, la última sentencia del programa se convertirá en:

```
Imprimir (lista, tamaño)
```

El procedimiento tendrá toda la información que necesita si le pasamos la lista de valores de datos y la cantidad de elementos, de modo que en este caso el encabezamiento no necesitará ninguna VAR:

```
PROCEDURE Imprimir (items  : ListaRegistro;
                    altura : limites);
```

Podríamos seguir adelante pasando a codificar todo el procedimiento, pero por ahora es mejor dejarlo como una "colilla" BEGIN...END y ocuparnos de los otros procedimientos de *nivel 1*. Hemos de elegir un nombre para cada uno, decidir qué elementos de datos es preciso pasar como parámetros a cada procedimiento, y determinar si es necesario pasar algunos de estos elementos como parámetros VAR (dirección).

Todos los métodos de estructuración de datos que hemos analizado hasta ahora han sido de tamaño fijo. Ello se especifica en las definiciones TYPE y, por consiguiente, se conoce en tiempo de compilación. El PASCAL proporciona estructuras de datos avanzadas cuyo tamaño puede variar (e incluso también su tipo, con ciertas restricciones) durante la ejecución de un programa. El tamaño de una estructura avanzada sólo está limitado por la memoria física o el almacenamiento de apoyo disponibles, y la estructura avanzada más familiar es el archivo secuencial.

Al igual que una matriz, cada elemento de un archivo puede ser de cualquier tipo, simple o estructurado, con la excepción de que uno no puede tener un archivo de archivos. Utilizando nuestra anterior definición de tipo de registro, podríamos añadir las siguientes declaraciones:

```
TYPE
    TipoArchivo=FILE OF datos;
VAR
    ArchivoDatos:TipoArchivo;
```

que nos permitirían crear y procesar un archivo que contuviera un número infinito de componentes, siendo cada uno de ellos un registro de un nombre, deuda y cualquier otro campo que deseáramos. Igual que decimos read (simbolo), queriendo significar leer un único char de la entrada del archivo, podemos decir: read (ArchivoDatos,item) o write (ArchivoDatos,item) y manipular toda una estructura de registro como un objeto de un solo dato. Si estos archivos sólo se requieren durante la ejecución del programa, la otra única exigencia consiste en abrirlos para leerlos o escribir en ellos mediante los procedimientos predefinidos reset y rewrite, respectivamente.

Si usted desea darles un carácter permanente, lo cual es más probable, entonces los identificadores de archivo deben especificarse en la lista de parámetros del encabezamiento del programa. En este caso, por ejemplo:

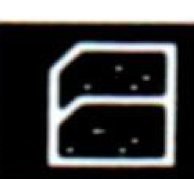
```
PROGRAM ManipuladorDatos (input,output,ArchivoDatos);
```

Cada vez que utilizamos sentencias read o write, llamamos a procedimientos de E/S de archivos predefinidos del PASCAL. Los dos únicos archivos que, en realidad, conocemos hasta ahora son los dispositivos de E/S estándares de nuestro ordenador. El archivo input normalmente es un teclado y output será, invariablemente, una pantalla VDU en los sistemas de microordenador. Al simular que estos dispositivos son archivos, en PASCAL la manipulación de todas las E/S resulta sumamente coherente. Recuerde que cuando decimos write(N), omitiendo cualquier nombre de archivo, el valor de N se imprime en forma de caracteres en el archivo output. Al omitir un nombre de archivo en sentencias read o ReadLn, se pasa por defecto al archivo input. Estos dos archivos son archivos de texto, es decir, cada "registro" es un valor de un único caracter.

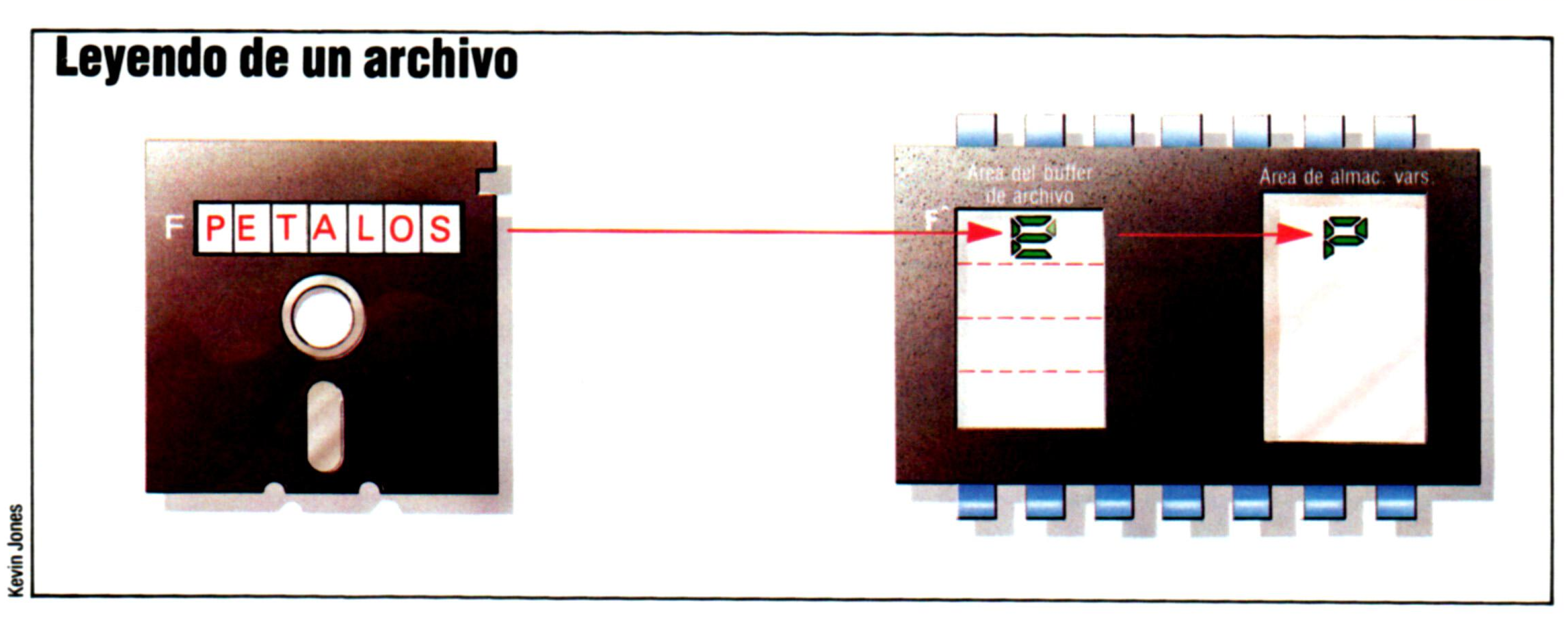
Archivos de texto

Sin embargo, a diferencia de otros archivos, los archivos de texto pueden tener (además de una marca de final de archivo) indicadores especiales de final de línea empotrados en cualquier punto dentro de ellos.

Estos indicadores varían con los diferentes sistemas operativos, y pueden constar de un solo carácter de control, de dos caracteres (CR y LF, p. ej.), o



Leyendo de un archivo



A la sombra de un archivo
El proceso de abrir un archivo F en PASCAL prepara una zona buffer asociada, F^, en la cual se lee el primer carácter del archivo. Cuando se realiza una lectura, el carácter de la zona del buffer se le asigna a una variable y se transfiere a la zona del buffer el siguiente carácter del archivo. De este modo, si los primeros caracteres de F fueran PETALOS, al abrir el archivo P se leería en F^. Tras la primera operación de lectura, P sería asignada a una variable del PASCAL, y sería reemplazada por E en el buffer, F^

tal vez de ningún carácter, almacenándose, en cambio, la longitud de cada línea.

Con el fin de preservar la portabilidad, el PASCAL proporciona la función EoLn (F) y los dos procedimientos predefinidos ReadLn(F) y WriteLn(F), los cuales sólo se pueden emplear con archivos de tipo text. La función EoF(F), por supuesto, se puede utilizar con cualquier tipo de archivo. Dado que el indicador de final de línea puede o no ser un único carácter, la lectura de un valor char cuando EoLn es verdadera devolverá un carácter Space. Por consiguiente, normalmente debemos comparar previamente con EoLn. Puesto que tanto input como output existen antes y después de la ejecución de cualquier programa, están permanentemente abiertos, y no es preciso cargarlos ni crearlos de forma explícita.

Con archivos distintos de input y output, debemos asignarlos a un nombre del sistema externo y luego abrirlos para la lectura con una llamada al procedimiento reset (p. ej., reset (AlgunArchivo)), o bien crear una entrada en el directorio preparatoria para escribir en ellos mediante rewrite (Otro Archivo). Por lo tanto, un esquema general para procesar un archivo de texto es simplemente:

```
{abrir el archivo}

WHILE {no se llegue al final del archivo fuente}DO
   WHILE {no se esté en el final de una línea}DO
      {leer un carácter}
      {procesar el carácter}
      {saltarse el final de la línea}
```

PUT y GET

Los procedimientos read y write en realidad se implementan utilizando buffers de archivos y las primitivas de E/S put (para salida) y get (para entrada desde archivos). Cuando se reescribe un archivo, en el buffer no se coloca ninguna información hasta que se lleva a cabo un write. Éste en realidad consta de las dos operaciones:

```
F^:=datos;
put (F)
```

Del mismo modo, la sentencia read (F,datos) también se puede expresar:

```
datos :=F^;
get (F)
```

Por lo tanto, las sentencias de E/S de nuestro procedimiento Copiar:

```
read (fuente,caracter);
write (destino, caracter)
```

se podrían haber codificado igualmente como:

```
destino^:=fuente^;
put (destino);
get (fuente)
```

De esta manera no necesitaríamos de la variable char local, caracter. Tal vez se podría alcanzar una mejor comprensión de la operación de ReadLn (F) si la expresáramos en términos de estas primitivas:

```
WHILE NOT EoLn (F) DO
   get (F); {descartar el resto de la línea, si lo
   hubiera}

get (F) {.. y saltar el/los caracter/es EoLn}
```

De modo, entonces, que tras una ReadLn el buffer del archivo siempre contendrá el primer elemento de la siguiente línea a leer. Éste podría ser un espacio si EoLn (F) fuera verdadera, o estar indefinido si EoF (F) fuera verdadera. Obviamente, es ilegal intentar leer un archivo cuando EoF es verdadera, pero también es un error llevar a cabo cualquier operación, incluyendo la comprobación del buffer del archivo. Ello significa que se debe ser cuidadoso al manejar archivos que no sean ni input ni output. Pero, al mismo tiempo, el tener que estar atento a condiciones de error potenciales tales como éstas favorecerá la escritura de un software eficaz.

Ahora que sabemos cómo "anticiparnos" a un flujo de datos venidero, podemos formular el procedimiento SaltarBlancos que proponíamos en el capítulo anterior.

```
PROCEDURE SaltarBlancos (VAR F : text);

  CONST
    espacio = ' ';
  VAR
    hecho : boolean;
  BEGIN
    hecho := EoF (F);

    IF NOT hecho THEN
      hecho := input^> espacio;
```

```
      WHILE NOT hecho DO
        BEGIN
          get (F);
          hecho := EoF (F);
          IF NOT hecho THEN
            hecho := F^> espacio
        END
    END; {SaltarBlancos}
```

Observe que así saltaremos todos los espacios en blanco de un archivo de texto, incluyendo los caracteres de final de línea. Si sólo deseáramos saltar espacios en una línea dada, podríamos alterar la asignación condicionada:

```
hecho := EoLn (F) OR (F^> espacio)
```

Si posteriormente deseáramos saltar todos los espacios en blanco:

```
REPEAT
  SaltarBlancos (F);
  IF NOT EoF (F) THEN
    TextoHallado := NOT EoLn (F)
  UNTIL TextoHallado OR EoF (F)
```

Con esta última modificación, resulta muy sencillo escribir programas interactivos que comprueben entradas nulas.

Por ejemplo:

```
REPEAT
  write ('Entre datos:');
  SaltarBlancos (input)
UNTIL NOT EoLn (input)
```

Esto fracasará en el caso de que se entre el carácter de control que utilice el sistema para indicar el final del archivo, pero siempre podríamos usar el esquema anterior si deseáramos conseguir que nuestro programa fuera absolutamente seguro.

El procedimiento assign ha llegado a considerarse como una *ampliación estándar* y podría ser que pronto se adoptara con carácter oficial, así como open y seek para archivos de acceso directo. Otras ampliaciones esenciales de uso común son:

```
FUNCTION Fstat (NombreArchivo)
```

que devuelve un resultado booleano: (true si ya existe el archivo), y una facilidad:

```
PROCEDURE Rename (NombreViejo, NombreNuevo)
```

Complementos al PASCAL

No existe restricción alguna sobre el número de archivos que es posible tener abiertos. Sin embargo, puesto que éstos requieren la utilización del OS, nos hallamos en un campo en el cual existen divergencias. Por ejemplo, algunas versiones del lenguaje no soportan archivos. Las convenciones del OS para la asignación de nombres a los archivos puede, por tanto, suponer un problema. Muchos PASCAL de ordenadores centrales toman los primeros 10 caracteres, más o menos, de todo identificador de archivo, y lo relacionan con un archivo que posea ese nombre, de modo que nuestro ejemplo de archivo permanente crearía uno llamado ARCHIVODATOS. En muchos sistemas, las convenciones para los nombres son tales que esto no es posible. Ejemplos: A CP/M-FIL.DAT, #4:APPLEFOR MAT, etc. La "ampliación estándar" para conectar un identificador de archivo es el procedimiento de asignación, empleado antes de reestablecer o reescribir:

```
assign (IdentificadorArchivo,SerieNombre)
```

De modo que, a nuestros fines actuales, bastaría assign (ArchivoDatos,D:NuestroArch.dat)

Programa CopiarTexto

El PASCAL proporciona el identificador "text" para el tipo más común de archivo, y éste se utiliza cuando declaramos todas las variables del archivo de texto en la lista de parámetros del encabezamiento del programa. Aquí ofrecemos una facilidad para copiar archivos de texto. Con el objeto de lograr que el programa sea útil con el carácter más general posible, hemos formulado todo el proceso de copiado como un procedimiento separado denominado Copiar, que puede procesar dos archivos de texto cualesquiera. Recuerde que los compiladores no estándares exigen la utilización de reset (F1,'Fuente'), etc., en el programa principal. Observe que ambos archivos se pasan al procedimiento Copiar como parámetros VAR. Ello se debe a que no sólo se actualiza el archivo de destino, sino que se lee el archivo fuente y, por lo tanto, cambia el estado del archivo. Por este motivo, las variables de archivo siempre se pasan por dirección, jamás por valor. Aparte de cualquier otra consideración, un parámetro de valor implica una copia local. Pasar grandes estructuras, tales como matrices de registros, también se puede considerar una excepción por razones de conservación de memoria

```
PROGRAM    CopiarTexto    (F1,F2);

VAR
    F1,
    F2         : text;

{11111111111111111111111111111111}

PROCEDURE  Copiar (VAR fuente,
                       destino   : text);
  VAR
    caracter             : char;
  BEGIN
    WHILE NOT EoF (fuente) DO
      BEGIN {copiar una linea:}
        WHILE NOT EoLn (fuente) DO
          BEGIN
            read (fuente, caracter);
            write (destino, caracter)
          END;
            {ahora copiar el final de linea:}
        ReadLn (fuente);
        WriteLn (destino)
      END
  END;   {Copiar}
{11111111111111111111111111111111}

BEGIN      {CopiarTexto — Programa principal}

  assign (F1, 'Fuente');
  reset (F1);              {localizar y abrir para lectura}
  assign (F2, 'Destino');
  rewrite (F2);            {crearlo}
  Copiar (F1, F2)

END.
```

Administrador eficaz

En esta ocasión concentraremos nuestra atención en el "MicroPen", DBM diseñado para el Amstrad CPC 464

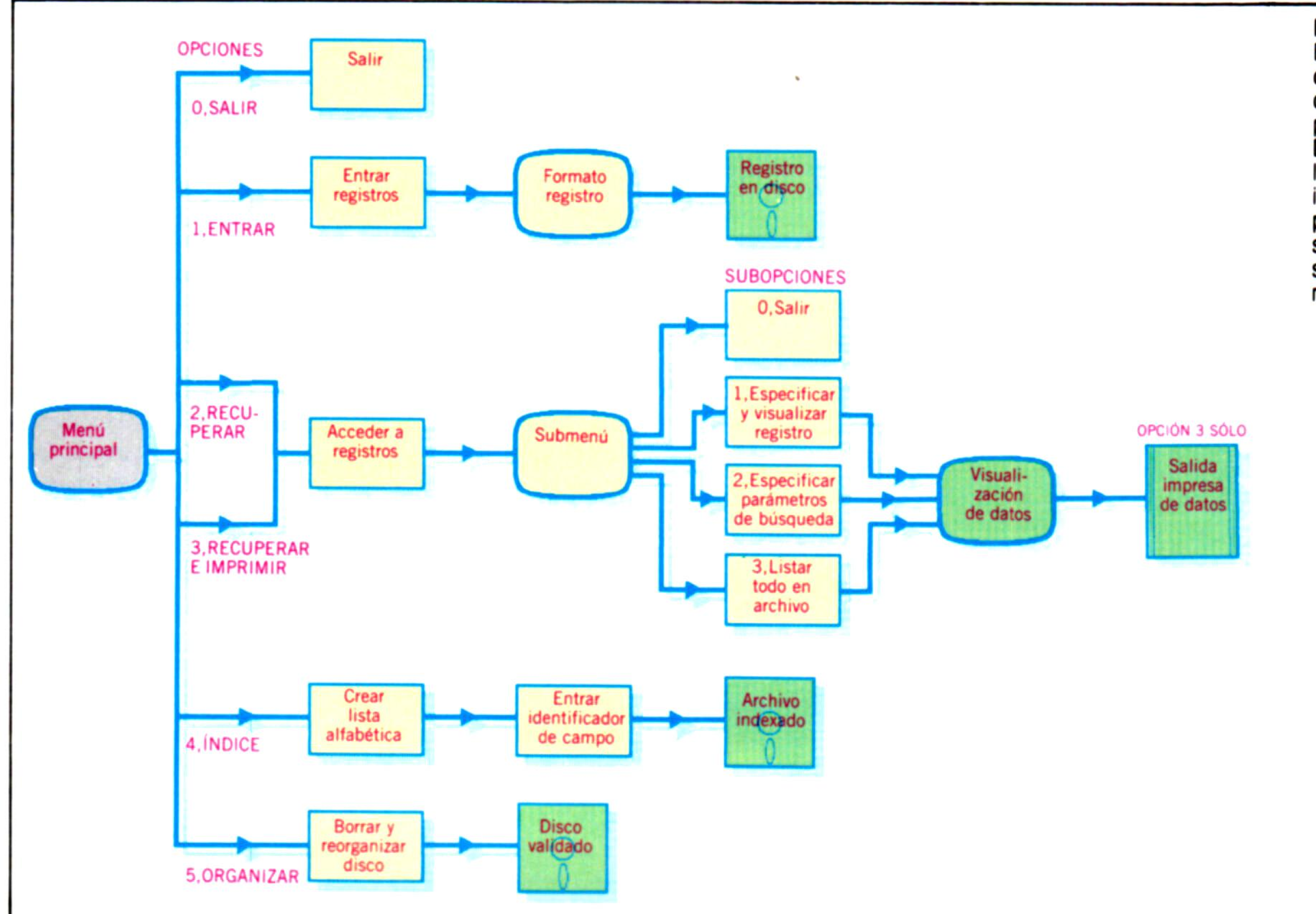
Descrito como un "sistema de archivo de bases de datos para el CPC 464", *MicroPen* está editado por la división Amsoft de Amstrad, pero es obra de la empresa de software Intelligence Ireland. *MicroPen* forma parte de un trío de programas que comprende un DBM, un procesador de textos y una hoja electrónica. Desde el punto de vista conceptual, el paquete guarda similitud con juegos como el *Lotus 1-2-3* y el *PIPS* de Sord. El componente de DBM incluye un procesador de textos denominado Penform. El mismo se utiliza para crear formatos en pantalla de los registros a utilizar dentro de un archivo DBM. El DBM propiamente dicho se denomina simplemente Pen.

La versión Amstrad del *MicroPen* funciona bajo CP/M-80 y, por consiguiente, requiere al menos una unidad de disco DDI-1. Aunque esto parece un gasto adicional algo desafortunado, usted comprenderá enseguida que la capacidad y velocidad de los discos es esencial; los DBM basados en cassette pueden resultar penosamente lentos y frustrantes.

Para crear un archivo de base de datos es necesario crear primero el trazado o formato para los registros que compondrán el archivo. Ello se realiza ejecutando el Penform. Si bien en la documentación se alude a éste como un procesador de textos, en realidad no es más que un editor en pantalla. Es decir, permite entrar o modificar caracteres en la pantalla, desplazando el cursor mediante las teclas para el mismo. Las teclas del cursor se complementan con instrucciones de edición simples, tales como [CTRL] Y para suprimir una línea o [ESC] E para salvar en disco. El Penform no es un auténtico procesador de textos, puesto que no se puede utilizar para crear y editar documentos completos como cartas o artículos.

Hay algunas limitaciones en cuanto a la cantidad de información que se puede incluir en un archivo de base de datos *MicroPen*. Un registro (denominado *layout* —trazado— en la documentación) puede tener hasta 100 campos, pero ningún registro puede contener más de 1 024 caracteres. Teóricamente, un archivo puede contener hasta 32 750 registros, pero en la práctica no es viable una cantidad tan alta de registros en un sistema basado en disco flexible. Los 32 750 registros completos, conteniendo cada uno de ellos los 1 024 caracteres admisibles como máximo, requerirían más de 33 megabytes de almacenamiento en disco sólo para los datos en bruto (suponiendo que no se utilicen técnicas para compresión de datos).

Una característica inusual del *MicroPen* es que a cada uno de los campos de un registro se le ha de otorgar un identificador exclusivo en pantalla.



Plan de acción
Éstas son las diferentes opciones y procedimientos de operador que ofrece el menú principal del *MicroPen*. Los DBM que poseen su propio lenguaje de programación incorporado, como el *dBase II*, permiten la construcción de secuencias de procedimiento similares según las propias necesidades

Estos identificadores se denominan *señaladores de campo* y pueden ser cualquier carácter imprimible a excepción del corchete, que no está disponible porque posee un papel especial como delimitador de campo.

Si estuviéramos utilizando el *MicroPen* como DBM para nuestra base de datos COMPONENTES, un trazado de registro creado por Penform tendría el siguiente aspecto:

Número de componente del fabricante: [A]
Número de componente nuestro: [B]
Precio: [C] Cantidad en stock: [D]
Descripción: [E]
Proveedor: [F]
Num. teléfono del proveedor: [G]
Hablar con: [H]

Tras haber creado un formato como éste, es guardado en disco empleando un nombre de archivo como COMPONENTES.INP (la "extensión" .INP del nombre del archivo la exige el programa Pen DBM).

Ejecución del "Pen"

Al ejecutarlo, el *Pen* visualiza un mensaje en la pantalla que pregunta qué archivo de base de datos se ha de utilizar. Para emplear nuestra base de datos de inventario, responderíamos COMPONENTES (no se requiere aquí la extensión .INP para el nombre del archivo). *Pen* visualiza entonces un menú, conocido como *menú principal*, que ofrece seis opciones:

Base de datos: COMPONENTES, por registro 328, registros en archivo 0.

0=Salir, 1=Entrar, 2=Recuperar, 3=Recuperar e imprimir, 4=Indice, 5=Organizar

Para entrar registros se necesita la opción 1. La línea superior (línea de estadística) señala que hay 328 caracteres por registro, pero que en el archivo no hay (todavía) ningún registro. Tras la selección de la opción 1 se visualiza en la pantalla el formato de registro con instrucciones. En nuestro ejemplo, en la pantalla se visualizaría:

Pulsar {ESC} tras completar entrada de datos para registro 1
Pulsar^C para borrar campo

Número de componente del fabricante:
Número de componente nuestro:
Precio: Cantidad en stock:
Descripción:
Proveedor:
Num. teléfono del proveedor:
Hablar con:

Los datos se digitan de la forma habitual. Tras entrar los datos para un campo, la pulsación de ENTER concluye la entrada para ese campo y desplaza el cursor hasta el campo siguiente. Los errores cometidos en un campo se pueden corregir mediante el empleo de la tecla Delete. Una vez entrado correctamente todo el registro, utilizando la tecla Escape se visualizará un submenú:

0=Salir
1=Continuar
2=Escribir registro en archivo

La opción 2 escribe el registro en disco y visualiza el formato para el siguiente registro.

La opción 2 del menú principal permite acceder a registros o recuperarlos y da lugar a otro submenú:

Recuperar por: 0=Salir
1=Número de registro
2=Búsqueda
3=Listar todo el archivo

Las opciones 0 y 1 se explican bastante por sí solas; la opción 1 permite especificar y visualizar un registro; da por sentado, no obstante, que usted conoce el número de registro que quiere, aunque es poco probable que usted lo recuerde si el archivo contiene muchos registros. La opción 2 permite especificar varios parámetros de búsqueda para poder localizar cierto registro que cumpla la especificación. El submenú de búsqueda es:

Pulsar Escape cuando el perfil de búsqueda esté completo. Para establecer modalidad de búsqueda: ^Q=Contiene,^W=No contiene,^E=Igual a,^R=No igual a, ^T=Mayor que,^Y=Menor que

Usted ya habrá percibido que hay una gran incoherencia en cuanto a las instrucciones y opciones de menú para las diversas partes del paquete: algunas veces debe pulsarse la tecla Escape, otras veces una combinación de CRTL–letra, otras un número de menú. Muy raramente las combinaciones CTRL–letra poseen algún verdadero valor mnemónico: ^E para hallar un registro con campos de comparación durante una búsqueda, ^Q para especificar Contiene, etc. Sea como fuere, las opciones de "búsqueda" permiten una razonable flexibilidad para especificar los registros que se han de buscar.

Habiendo dado los parámetros que definen el registro requerido, las combinaciones CTRL–letra permiten localizar registros que contengan los datos especificados, que no contengan los datos especificados, que posean campos con datos emparejados, que no posean campos con datos emparejados, o que posean un valor de campo menor que el valor de campo especificado. Si quisiera, por ejemplo, localizar registros de componentes con un PRECIO inferior a 37,50, lo podría conseguir con gran facilidad.

MicroPen puede crear un índice de todas las entradas de un archivo de base de datos. La indexación se efectúa por campo, y debe conocerse también la cantidad máxima de registros. El campo se especifica por su identificador, no por el nombre que le hemos dado al campo. Para obtener un índice de **Número de componente nuestro**, y suponiendo que en el archivo hubiera 500 registros, especificaríamos B=#500 (siendo B el identificador para nuestro campo **Número de componente nuestro**).

MicroPen ofrece capacidades de búsqueda e indexación moderadamente avanzadas, y por su precio es asequible a muchos usuarios de ordenadores personales. No ofrece, sin embargo, la ventaja de un lenguaje de programación incorporado, como los que incluyen *Archive* y *dBase II*. Aunque este último es un paquete caro diseñado para ordenadores caros, *Archive* viene "gratis" con el Sinclair QL y a un precio global comparable al de un Amstrad CPC 464 más unidad de disco y software *MicroPen*. Ésta es la clase de consideraciones a tener presentes antes de adquirir un sistema de ordenador con el software de DBM asociado.



Ampliación Amstrad

El paquete de unidad de disco DDI-1 de Amstrad proporciona una mejora a buen precio para la máquina

El concepto del ordenador como un artículo más de la electrónica doméstica, como un televisor o un equipo de alta fidelidad, instalado con carácter permanente en un rincón de la habitación en vez de tenerlo que montar y conectar a otros componentes cada vez que se desea utilizarlo, fue obviamente recibido con satisfacción por los entusiastas. Existe, asimismo, la ventaja de evitar conflictos con quienes desean usar el televisor.

El Amstrad es muy popular entre los usuarios personales, pero posee un inconveniente fundamental: a pesar de que la empresa incorporó en la máquina una unidad de cassette, el ordenador carecía de una capacidad de almacenamiento rápido. En el momento del lanzamiento se prometió una unidad de disco, lo que llevó a mucha gente a adquirir el ordenador a la espera de que la misma saliera rápidamente a la venta. Sin embargo, las remesas no aparecieron hasta el año siguiente y desde entonces la adquisición del dispositivo se puede realizar cada vez con más facilidad a través de las tiendas minoristas.

El paquete de unidad de disco se compone de la unidad de disco, una interface (que permite conectar la máquina a la puerta para disco flexible de la parte posterior del ordenador), un disco de sistema y un manual. El otro extremo del cable de la interface se desliza en el conector de 34 vías de la parte posterior de la unidad de disco.

La unidad de disco propiamente dicha utiliza discos de 3 pulgadas de estándar Hitachi. Ésta parece una elección extraña, ya que es Sony y no Hitachi quien aparece como probable ganadora de la batalla por captar el mercado de microflexibles. El formato Sony de 3 ½ pulgadas parece estar a punto de convertirse en el estándar, puesto que un creciente número de fabricantes, incluyendo Apple, Apricot y, más recientemente, Acorn con su Electron, han adoptado los discos Sony para sus unidades. A excepción de Amstrad, ningún otro de los grandes fabricantes de ordenadores parece haber optado por los discos Hitachi.

En el interior de la unidad, dos motores claramente visibles controlan la rotación del disco y el cabezal de lectura/escritura. En la parte posterior está la fuente de alimentación, separada del mecanismo de la unidad de disco mediante una placa metálica que la protege del exceso de calor y los campos magnéticos. En la parte posterior de la carcasa y encima de la puerta para la interface hay un interruptor de on/off.

Los discos utilizados en el Amstrad son similares, en concepto, a sus equivalentes Sony, aunque su aspecto es muy diferente. Los discos Amstrad miden 100×80×4 mm y, al igual que los discos Sony, se hallan dentro de una carcasa plástica y poseen una placa metálica sobre la ventana de lectura/escritura que protege al disco de las huellas dactilares o la suciedad. Ésta se retrae cuando el disco se coloca en la unidad. Los discos Amstrad, sin embargo, poseen sus escudos en el interior de la carcasa metálica, mientras que Sony los ha colocado en el exterior.

La unidad operó de forma rápida y fiable y no tuvimos ningún problema para hallar archivos y cargarlos en unos pocos segundos, pero pareció más ruidosa que el producto de Sony. No obstante, resultó más silenciosa que la media de unidades de disco flexible de 5 ¼ pulgadas promedio.



Chris Stevens

Escalando el mercado
La unidad de disco Amstrad DDI-1 permite que los usuarios amplíen sus sistemas colocándolos a la altura de las especificaciones que requiere un microordenador "serio". Con la adición de los tres programas que se suministran junto con la unidad de disco (CP/M, AmsDOS y Dr LOGO), el Amstrad CPC 464 posee ahora un número mucho mayor de aplicaciones. La unidad se conecta al ordenador mediante el cable de interface que se proporciona, a través de la puerta interface para disco flexible

El disco del sistema

El disco del sistema contiene tres utilidades básicas: AmsDOS, el propio sistema operativo de disco de Amstrad, y dos utilidades de Digital Research: el sistema operativo de disco CP/M, tan ampliamente utilizado, y Dr LOGO, una popular implementación del lenguaje de aprendizaje y de gráficos tortuga que cada vez se está extendiendo más, con evidencias incluso de desplazar al BASIC como lenguaje principal en los ordenadores personales.

Refiriéndonos en primer lugar al AmsDOS, quizá ésta sea la más floja de las tres utilidades proporcionadas. Para poder ejecutar el AmsDOS, debe primero cargarse el CP/M. Esto tal vez parezca extraño, pero el resultado es que el AmsDOS utiliza igual memoria que el CP/M solo y, por tanto, el CP/M se desecha tras haber cargado al AmsDOS. A diferencia del CP/M, que es un siste-

ma operativo autocontenido, el AmsDOS simplemente añade instrucciones de operación de disco al carácter barra vertical ❙ (Shift @); por ejemplo: ❙ DRIVE y ❙ DIR.

El procedimiento para manipular archivos en disco desde AmsDOS es uno de los más peculiares que puedan verse en cualquier micro personal. Por ejemplo, para borrar (ERA*se*) un archivo, usted debe primero asignar el nombre del archivo a una serie; sólo entonces podrá suprimir el archivo mediante el borrado de la serie. Por consiguiente, para borrar el archivo FACTURA.$$$, la secuencia de instrucciones sería:

```
A$="FACTURA.$$$"
❙ ERA,aA$
```

Este método se complica aún más al renombrar un archivo (REN*ame*), dado que dos archivos diferentes (nombre viejo y nombre nuevo) se han de asignar a series que se puedan manipular. A muchos usuarios les resultará más sencillo cargar el CP/M para tales procedimientos, que se pueden llevar a cabo mediante una única línea. No obstante, el CP/M posee sus propias dificultades. Fundamentalmente, no hay facilidades para ejecutar el BASIC Amstrad bajo CP/M, de modo que usted se verá obligado a utilizar el AmsDOS si desea programar el sistema de unidad de disco en BASIC.

El sistema operativo CP/M del disco de sistema es la versión 2.2 que se ha venido aplicando en gran número de máquinas de gestión de ocho bits en la última década. Junto con el propio CP/M se proporcionan en el sistema las instrucciones "transitorias" usuales. Las instrucciones transitorias son aquellas que están retenidas permanentemente en disco y que se cargan en la memoria del ordenador sólo cuando se las necesita; luego son desechadas por el sistema operativo CP/M. Aquí el problema es que para poder utilizar una gran cantidad de instrucciones en CP/M (como PIP, que transfiere un archivo de un dispositivo periférico a otro), el sistema operativo realmente requiere que haya dos discos instalados en el ordenador, uno para retener el disco de sistema de modo que se pueda acceder rápida y sencillamente a las instrucciones transitorias, y otro para retener los archivos.

Aparte de la necesidad de utilizar el CP/M para intercambiar continuamente los discos de datos y de sistema, poniéndolos y sacándolos en una única unidad, muchas instrucciones no tienen en cuenta que el usuario tenga una sola unidad de disco disponible y esperan que pueda transferir los archivos de una unidad a otra. Para contrarrestar esto, Amstrad ha incluido instrucciones transitorias adicionales para la transferencia de archivos con una sola unidad. FILECOPY proporciona mensajes que permiten el intercambio de discos al transferir un archivo y, de forma similar, DISCCOPY copia un disco completo. Aun así, usted habrá de adquirir dos unidades para sacar el máximo partido del sistema.

Puede decirse, no sin cinismo, que la de Amstrad es una maniobra inteligente para obligar a los clientes a adquirir dos unidades de disco en lugar de una. Sin embargo, una explicación más benévola es que el suministrar el CP/M tiene su sentido en función de la estrategia a largo plazo de Amstrad. La filosofía que subyace tras la gama de ordenadores de la empresa no es la de crear productos de tecnología punta, sino la de proporcionar equipos ya probados y comprobados que se puedan utilizar para una amplia gama de aplicaciones. De allí la decisión de la empresa de optar por el procesador Z80, cuando la mayoría de los fabricantes se disputan por producir máquinas de 16 bits. El empleo del CP/M encaja bien, puesto que, aunque no sea el sistema operativo de disco más amable con el usuario, es adaptable y en todo el mundo hay miles de programadores que poseen la experiencia necesaria para escribir software para ejecutar bajo el sistema.

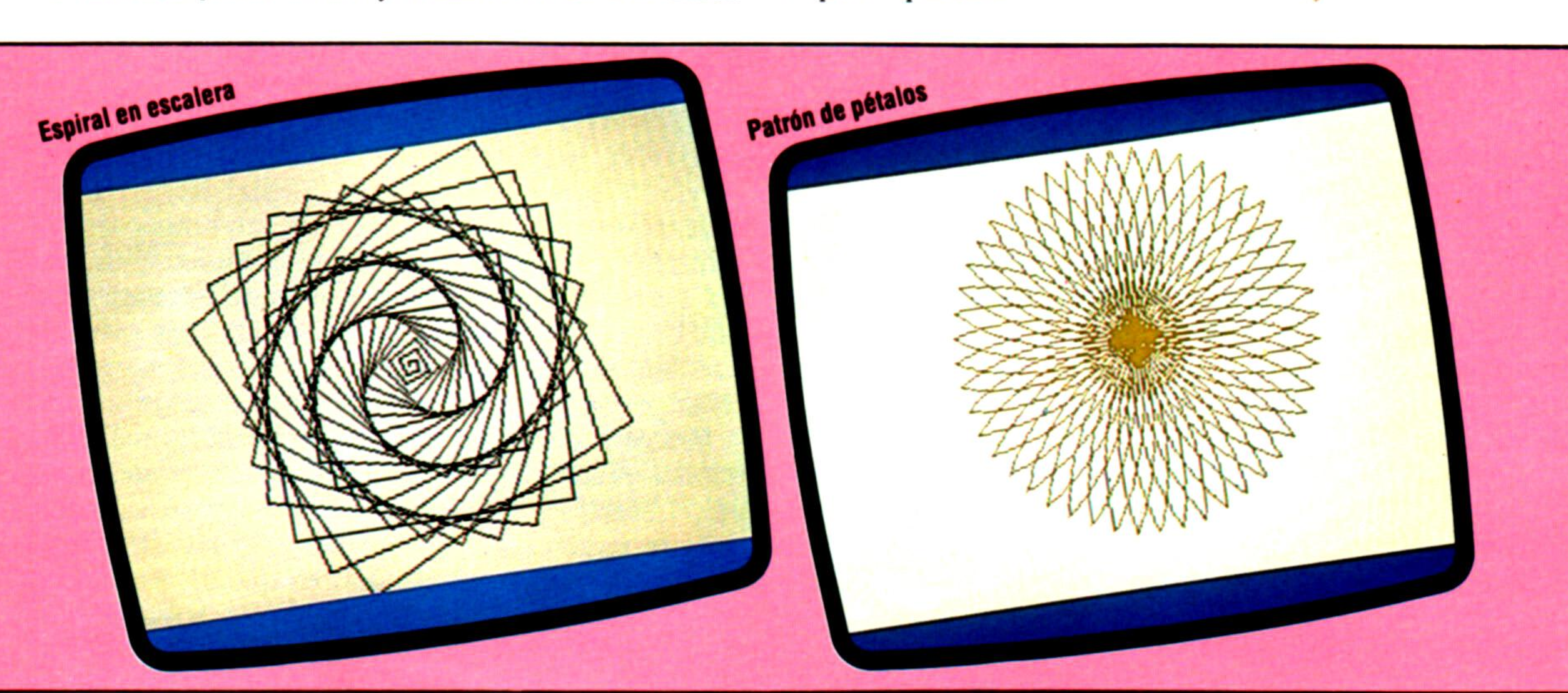
A pesar de los problemas que supone la utilización del CP/M con una sola unidad de disco disponible, Amstrad merece felicitaciones por proporcionar una implementación completa de un OS de disco tan potente como éste en un micro personal.

El LOGO de Digital Research

Dr LOGO es una versión del popular lenguaje que se utiliza en muchos centros educativos europeos. El lenguaje opera bajo CP/M, que, por consiguiente, se ha de cargar antes de poder cargar el LOGO, y el Dr LOGO puede aprovechar las facilidades operativas de disco ampliadas del CP/M. Por ejemplo, el CP/M permite que los archivos se puedan llamar sin tener que digitar un nombre completo; de modo que, digitando un nombre parcial, el lenguaje llamará a todos los archivos que encajen con esa descripción parcial.

Arte de tortuga

Estos patrones se crearon utilizando el lenguaje Dr LOGO. Los procedimientos que empleamos para crearlos sólo exigieron unas pocas líneas de código cada uno. Aunque el Dr LOGO permite alterar los colores de fondo y primer plano desde el lenguaje, no se ha previsto la modificación de la forma de la tortuga ni el coloreado de las formas que se dibujan. No obstante, desde el LOGO se puede generar sonido





AMSTRAD DDI-1

DIMENSIONES
280×105×75 mm

CAPACIDAD
Disco de sistema: 169 K; disco de datos: 178 K

INTERFACES
Interface individual en paralelo de 34 vías. El cable de interface tiene capacidad para dos unidades de disco

DOCUMENTACION
El manual que se proporciona es muy irregular; en ocasiones resulta difícil localizar la información que se desea

VENTAJAS
La provisión de CP/M y del Dr LOGO coloca al ordenador Amstrad al nivel de una máquina muy potente

DESVENTAJAS
El AmsDOS no está muy bien implementado y los usuarios que dispongan de una sola unidad encontrarán que ni el AmsDOS ni el CP/M son totalmente adecuados para sus necesidades

Cabezal de lectura/ escritura
El cabezal de lectura/escritura detecta los cambios que se producen en el campo magnético del disco y los traduce a señales eléctricas

Motores
Estos motores se utilizan para alimentar la unidad. El motor de abajo hace girar el disco, mientras que el otro mueve el cabezal de lectura/escritura a la pista correcta

Ranura para disco
Aquí se insertan los discos Hitachi de 3 pulgadas

Fuente de alimentación
El DDI-1 posee su propia fuente de alimentación eléctrica incorporada. Está separada del mecanismo de la unidad mediante una placa metálica. Ésta disipa el calor que pudiera generarse dentro de la fuente de alimentación y la protege de los campos magnéticos

Placa de circuitos
Regula la fuente de alimentación de modo que los motores eléctricos se activen en todo momento a la velocidad adecuada

Chris Stevens

El LOGO permite emplear las capacidades de sonido y gráficos del Amstrad y la implementación les resultará familiar a quienes hayan seguido nuestra serie dedicada al LOGO. También se ha previsto que el lenguaje sea operado mediante una palanca de mando y pulsadores de disparo, de modo que es posible escribir juegos en LOGO que se puedan desarrollar fácilmente mediante control por palanca de mando.

Si bien la implementación del Dr LOGO es muy buena y sale bien parada en comparación con otras muchas versiones del lenguaje, existen ciertas dudas sobre cómo se desempeñará bajo las limitaciones del hardware. Por ejemplo, un programa de teselado (uno que exige que el mismo diseño se repita varias veces con el fin de llenar la pantalla) que funciona a la perfección en el Commodore 64, generó en el Amstrad un mensaje "no hay suficiente espacio en la pila", lo que indica que quizá el lenguaje no quepa tan cómodamente como debiera. Aparte de esto, el Dr LOGO es rápido y amable y es una introducción ideal al lenguaje.

Tal como sucede con todos los sistemas de disco para micros personales, la popularidad del sistema, al menos al principio, dependerá del apoyo de software. Amstrad ya ha realizado numerosas utilidades para la unidad de disco, la mayoría de las cuales parecen dirigidas al aficionado personal más "serio". Estos paquetes incluyen una base de datos, un procesador de textos y lenguajes adicionales como PASCAL.

Al lanzar la unidad de disco para su máquina, Amstrad ha dado un paso evidentemente exitoso para hacer de su ordenador una máquina más acorde con el mercado sin imponer el precio que normalmente está asociado a tales sistemas. La adición de una unidad de disco ortoga al ordenador aún más competitividad y refleja las intenciones de Amstrad de ensanchar el horizonte de la máquina más allá del desfalleciente mercado de juegos.

Asegurando las escotillas

Prosiguiendo con nuestro proyecto de programación, nos ocuparemos del código de los cuatro restantes eventos mayores que se desarrollan antes de llegar a destino

El programa es dirigido al azar hacia una subrutina de contingencia mayor mediante la subrutina de la línea 6500, que genera un número aleatorio y lo utiliza en la sentencia ON X GOSUB de la línea 6510, descrita en el módulo anterior. Para dar cabida a la cantidad de posibilidades de este módulo, es necesario añadir a esta línea los primeros números de línea de las cuatro contingencias extras. La línea 6510 debe rezar ahora:

```
6510 ON X GOSUB 6530,6700,6800,6900,7000,
7050
```

El tercer número de línea llama a una subrutina en la cual el barco es atacado por piratas. El programa comprueba si esta contingencia ya se ha producido examinando el valor del indicador de eventos para esta subrutina, M(3). Si M(3) es 1, retorna al programa principal; de lo contrario continúa, poniéndolo a 1 en la línea 6818 para evitar la repetición.

Es posible que todos los tripulantes estén muertos, razón que haría irrelevante el ataque de los piratas. El programa prepara un bucle para comprobar los valores de la matriz de fortaleza de la tripulación, y cuenta los tripulantes con fortalezas de 0 o −999 examinando la tasa de fortaleza, TS(T,2), la matriz de fortaleza/categoría. Los tripulantes fallecidos o inexistentes se cuentan en X, y si X es igual a 16 no queda ninguno y el control se devuelve al programa principal.

Si la contingencia no se ha implementado con anterioridad y si quedan tripulantes a bordo, los piratas atacan, matando a algunos de ellos. La cantidad de muertos dependerá de que la tripulación disponga de armas para defenderse. La cuenta de los muertos se registrará en K, siendo dos el número mínimo. La línea 6825 establece el valor de K en esta cifra, y el programa examina entonces el elemento apropiado de la matriz de provisiones, OA(2), para ver si hay algún arma a bordo. Si el valor de OA(2) es 0 o −999, no hay ningún arma y K se establece en 4. Si hay armas a bordo, la variable S$ se establece en A PESAR DE TUS ARMAS. Si K se hubiera establecido en 4, S$ cambia por NO TIENES ARMAS.

Entre las líneas 6836 y 6845 se prepara otro bucle para suprimir el número adecuado de tripulantes, y la línea 6835 iguala X a 0 para conservar una cuenta de esta cifra. Si el valor de fortaleza de un determinado elemento es 0 o −999, el bucle pasa a considerar el siguiente elemento; si no lo es, la línea 6840 establece el valor en −999 e incrementa el valor de X en 1. Cuando el valor de X resulta igual al número de tripulantes que serán muertos, K, la línea 6842 establece el contador del bucle, T, en 16 y el programa sale del bucle. Si no quedan K tripulantes que matar, el programa saldrá del bucle tras efectuar la búsqueda 16 veces. Se imprime el número de tripulantes muertos, para completar la frase iniciada ya sea en la línea 6828 o bien en la 6830. Se solicita entonces al jugador que pulse cualquier tecla para continuar.

Rotura del timón

El siguiente acontecimiento consiste en la rotura del timón, que hace necesaria una reparación. Si se ha contratado algún mecánico, y éste está aún con vida, el timón se reparará rápidamente y el viaje se reanudará tras una breve demora; pero si no hay ningún mecánico a bordo, la travesía, como es lógico, durará más. La subrutina de rotura del timón comienza en la línea 6900 y es el cuarto número de línea de la sentencia ON X GOSUB de la línea 6510. Nuevamente, la subrutina comprueba si este evento ya se ha producido mediante el método usual: ver si el valor de M(4) se ha establecido en 1 y, si no fuera así, establecerlo en 1.

El valor de X se establece en 4, indicando la cantidad de semanas extras que durará el viaje si no hay ningún mecánico disponible. El programa revisa entonces la matriz de fortaleza/categoría, TS(,), en busca de un mecánico vivo. Entre las líneas 6930 y 6938 se establece un bucle que busca el número 3 en la matriz de categoría de tripulación de TS(,), que representa un mecánico, y un valor para la fortaleza del mecánico que no sea igual a 0 ni a −999, lo que significaría que el mecánico estaría vivo. Si se satisfacen todas estas condiciones, entonces el valor de X se restablece de 4 a 1. De haber disponible un mecánico la avería del timón se reparará enseguida y la duración del viaje se incrementará en sólo una semana.

No obstante, se le dice al jugador: AUNQUE TIENES UN MECANICO o, si X es igual a 4, NO TIENES NINGUN MECANICO Y. Entonces se imprime la duración extra de la travesía. Este valor se suma a la duración total del viaje, EW, en la línea 6965 y en este momento el juego continuará pulsando cualquier tecla. La variable X tiene, por consiguiente, dos finalidades: primero, indica el número de semanas a sumar al viaje, pero también actúa a modo de bandera, indicando si hay o no un mecánico disponible.

Una tormenta, manipulada por la subrutina de la línea 7000, constituye el quinto evento mayor posible. Desvía al barco de su curso y aumenta la duración del viaje. Si entre la tripulación hay algún oficial, el barco recuperará pronto el rumbo correcto.

Ian McKinnell

De no ser así, el viaje durará más tiempo. La estructura de esta subrutina es similar a la subrutina del "timón roto" y, de hecho, utiliza parte de su código.

El programa comprueba si la tormenta ya se ha producido y, en caso negativo, establece a 1 el valor de M(5). La duración extra del viaje se registra en el valor de X, que se establece en 2, y si no hay ningún oficial a bordo la travesía dura dos semanas más. Entre las líneas 7030 y 7038 se prepara un bucle para explorar la matriz de la tripulación en busca de un oficial, buscando un valor de 4 en la matriz de categoría, y un valor mayor que 0 en la matriz de fortaleza. Si se satisfacen estas exigencias, el valor de X se establece en 1, y T se establece en 16, haciendo que el programa salga del bucle.

S$ se establece en A PESAR DE QUE CUENTAS CON UN OFICIAL o en NO TIENES NINGUN OFICIAL Y, según el valor de X. El programa pasa entonces a la línea 6950, de la subrutina de rotura del timón, para utilizar de nuevo la sección del código que imprime la duración extra de la travesía, incrementa EW, la duración del viaje, en X, el tiempo extra, y retorna al programa principal.

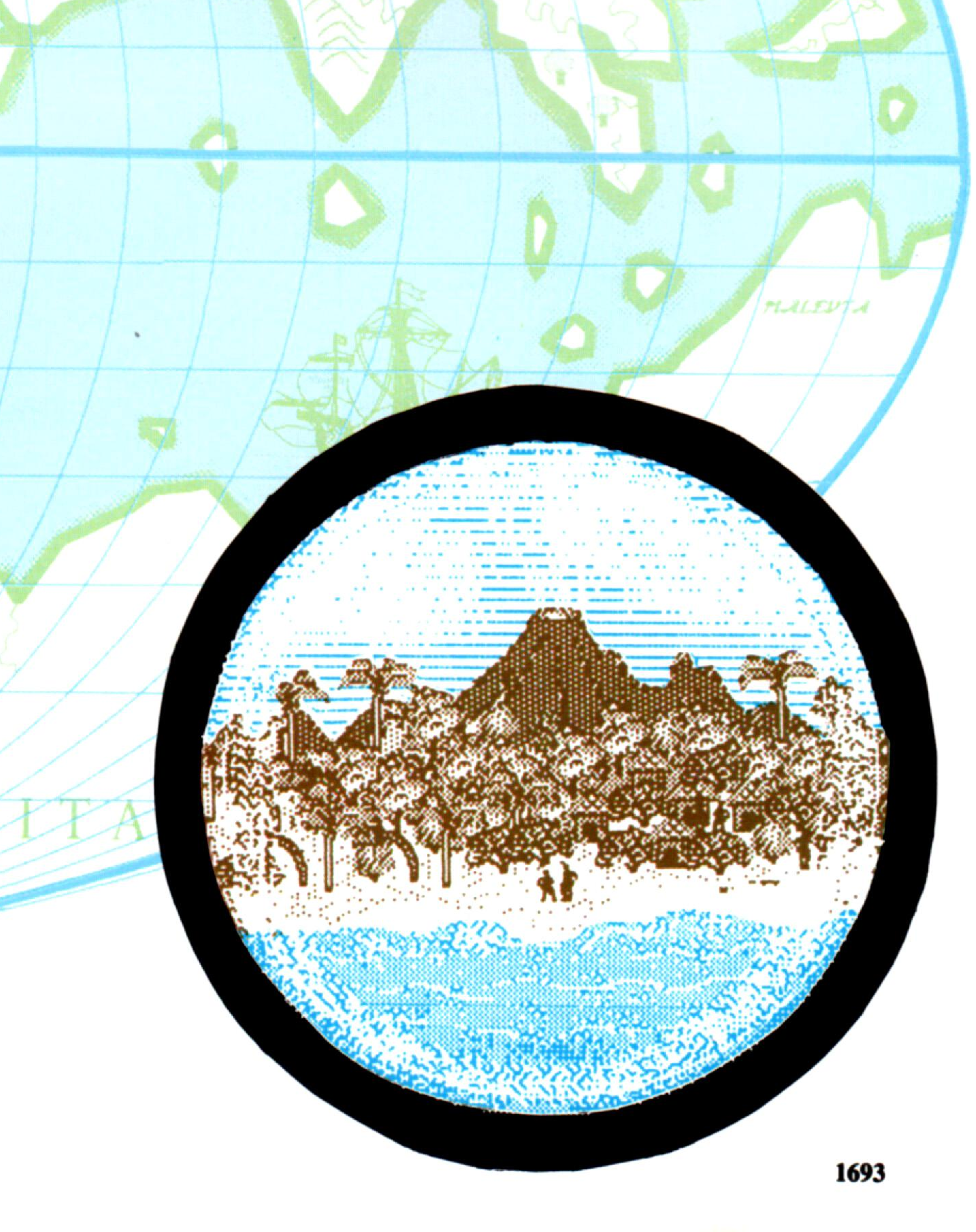
¡Tierra, tierra!

El último de los acontecimientos mayores, avistar una isla, se produce en la subrutina que empieza en la lína 7050. Éste es el último número de línea de la sentencia ON X GOSUB de la línea 6510. La isla no se halla en la ruta principal y, por tanto, una visita a la misma prolongaría la duración del viaje. No obstante, cambiando el rumbo y desembarcando, el barco podría reabastecerse de algunas provisiones. Por consiguiente, el jugador debe tomar la decisión de poner rumbo a tierra o continuar su recorrido actual. Si el barco visita la isla, el resultado de la expedición en busca de agua y comida podría verse fuertemente influido por cualquier encuentro previo con el albatros. ¡Si el jugador hubiera abatido al albatros el desvío no tendría ningún éxito!

El programa comprueba si la subrutina ya se ha ejecutado previamente, de la forma habitual. Se le informa entonces al jugador que las cartas indican la existencia de una isla en donde se podría reabastecer de provisiones y que el desvío prolongaría la duración del viaje, y se le pregunta si va a realizar la visita o no. La línea 7082 aguarda una respuesta y analiza el primer carácter de la entrada para determinar si la respuesta es sí o no. Si el jugador no desea alejarse de su rumbo, el control retorna al programa principal mediante la línea 7086. Si la respuesta es SI o S, el jugador arriba a la isla en la línea 7100.

El programa comprueba entonces si el jugador ha abatido al albatros, en la línea 7106. Si el albatros no fue muerto, B$, que se estableció en N en la línea 48, no se modificará y el programa pasará a la línea 7110. Sin embargo, si el jugador abatió al ave, B$ se habrá establecido en S mediante la línea 6162 de la subrutina del albatros. Si el ave fue muerta, la isla será una tierra yerma y el agua estará envenenada, no se conseguirán provisiones y se le recordará al jugador la circunstancia del disparo. El programa es enviado hasta la línea 7130, que incrementa la duración del viaje.

Si el albatros no fue derribado, en la línea 7110 se prepara un bucle de 1 a 4 para cada tipo de provisión, que recogerá las provisiones extras; la cantidad de suministros se selecciona al azar. La línea 7112 genera un número aleatorio y, si éste es menor que .25, pasa a la provisión siguiente. (Existe una posibilidad entre cuatro de no conseguir cada uno de los tipos de provisiones.) La línea 7115 genera un número aleatorio entre 5 y 14, almacenado en X. La línea 7120 imprime esta cantidad, seguida por las unidades de esa provisión en particular, ya sea en kilos o en barriles. Si durante la semana actual alguna provisión hubiera sido arrastrada por la borda, en la matriz de provisiones la cantidad se habría establecido en −999. De ser así, la línea 7122 restablece el valor a 0 para permitir la adición de las provisiones extras. Las provisiones extras, X, se suman a las provisiones existentes mediante la línea 7125. La travesía se incrementará en una o dos semanas. La línea 7135 decide de forma aleatoria cuál será el incremento, se le informa al jugador y se añade el tiempo extra a la variable de duración del viaje, EW, en la línea 7140.



Módulo 9: Otros eventos mayores

Inicializar banderas

```
48 A$="N":B$="N"
```

Rutina Contingencia de piratas

```
6800 REM PIRATAS
6805 IF M(3)=1 THEN RETURN
6810 X=0
6812 FOR T=1 TO 16
6814 IF TS(T,2)=0 OR TS(T,2)=-999 THEN X=X+1
6815 NEXT
6816 IF X=16 THEN RETURN
6818 M(3)=1
6820 PRINT CHR$(147)
6822 S$=" EL BARCO ES ATACADO POR PIRATAS!*":GOSUB 9100
6824 PRINT:GOSUB 9200
6825 K=2
6826 IF OA(2)=0 OR OA(2)=-999 THEN K=4
6828 S$="A PESAR DE TUS ARMAS*"
6830 IF K=4 THEN S$="NO TIENES ARMAS *"
6832 GOSUB 9100
6835 X=0
6836 FOR T=1 TO 16
6838 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6845
6840 X=X+1:TS(T,2)=-999
6842 IF X=K THEN T=16
6845 NEXT
6850 PRINT X;
6855 S$="TRIPULANTE HA RESULTADO MUERTO*"
6856 IF X>1 THEN S$="TRIPULANTES HAN RESULTADO MUERTOS*"
6860 GOSUB 9100
6865 PRINT:GOSUB 9200
6890 S$=K$:GOSUB 9100
6895 GET I$:IF I$="" THEN 6895
6899 RETURN
```

Rutina Contingencia del timón

```
6900 REM TIMON
6905 IF M(4)=1 THEN RETURN
6910 PRINT CHR$(147)
6915 M(4)=1
6920 S$="HAY PROBLEMAS CON EL TIMON!*":GOSUB 9100
6925 PRINT:GOSUB 9200
6928 X=4
6930 FOR T=1 TO 16
6935 IF TS(T,1)=3 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     X=1:T=16
6938 NEXT
6940 S$="A PESAR DE QUE TIENES UN MECANICO*"
6945 IF X=4 THEN S$="NO TIENES MECANICO Y*"
6950 GOSUB 9100
6955 S$="TU VIAJE DURARA*":GOSUB 9100
6960 PRINT X;"SEMANAS MAS"
6965 EW=EW+X
6967 PRINT:GOSUB 9200
6969 S$=K$:GOSUB 9100
6970 GET I$:IF I$="" THEN 6970
6975 RETURN
```

Rutina Contingencia de la tormenta

```
7000 REM TORMENTA
7005 IF M(5)=1 THEN RETURN
7010 PRINTCHR$(147)
7015 M(5)=1
7020 S$="EL VIENTO TE APARTA DE TU RUMBO*":GOSUB 9100
7022 S$="DURANTE UNA TORMENTA!*":GOSUB 9100
7025 PRINT:GOSUB 9200
7028 X=2
7030 FOR T=1 TO 16
7035 IF TS(T,1)=4 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     X=1:T=16
7038 NEXT
7040 S$="A PESAR DE QUE CUENTAS CON UN OFICIAL*"
7045 IF X=2 THEN S$="NO TIENES NINGUN OFICIAL Y*"
7049 GOTO 6950
```

Rutina Contingencia de la isla

```
7050 REM ISLA
7055 IF M(6)=1 THEN RETURN
7060 PRINTCHR$(147)
7065 M(6)=1
7070 S$="TUS CARTAS INDICAN LA EXISTENCIA DE UNA ISLA*":GOSUB
     9100
7071 S$="EN LA QUE PODRIAS*":GOSUB 9100
7072 S$="REABASTECERTE DE PROVISIONES*":GOSUB 9100
7073 S$="PERO SI TE DESVIAS HACIA ELLA*":GOSUB 9100
7074 S$="SUPONDRA UNA MAYOR DURACION DEL VIAJE*":GOSUB 9100
7075 PRINT:GOSUB 9200
7080 S$="QUIERES IR A LA ISLA?*":GOSUB 9100
7082 INPUT I$:I$=LEFT$(I$,1)
7084 IF I$<>"S" AND I$<>"N" THEN 7082
7086 PRINT:GOSUB 9200
7090 IF I$="N" THEN 7145
7100 S$="LLEGAS A LA ISLA*":GOSUB 9100
7105 S$="Y CONSIGUES*":GOSUB 9100
7106 IF B$="N" THEN 7110
7107 PRINT:GOSUB 9200
7108 PRINT"NADA!":GOSUB 9200
7109 S$="(RECUERDA EL ALBATROS!)":GOSUB 9100;GOTO 7130
7110 FOR T=1 TO 4
7112 IF RND(1)<.25 THEN 7129
7115 X=INT(RND(1)*10)+5
7120 PRINT X;U$(T);"S DE";P$(T)
7122 IF PA(T)=-999 THEN PA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="PERO AHORA EL VIAJE DURARA*":GOSUB 9100
7135 X=INT(RND(1)*2)+1
7139 PRINT X;:S$="SEMANAS MAS*":GOSUB 9100
7140 EW=EW+X
7145 PRINT:GOSUB 9200
7150 S$=K$:GOSUB 9100
7155 GET I$:IF I$="" THEN 7155
7159 RETURN
```

Complementos al BASIC

Spectrum:
Introducir las siguientes modificaciones:

```
6512 IF X=3 THEN GOSUB 6800
6513 IF X=4 THEN GOSUB 6900
6514 IF X=5 THEN GOSUB 7000
6515 IF X=6 THEN GOSUB 7050

6820 CLS
6895 LET I$=INKEY$:IF I$=""THEN GO TO 6895
6910 CLS
6970 LET I$=INKEY$:IF I$=""THEN GO TO 6970
7010 CLS
7060 CLS
7082 INPUT I$:LET I$=I$(1 TO 1)
7155 LET I$=INKEY$:IF I$=""THEN GO TO 7155
```

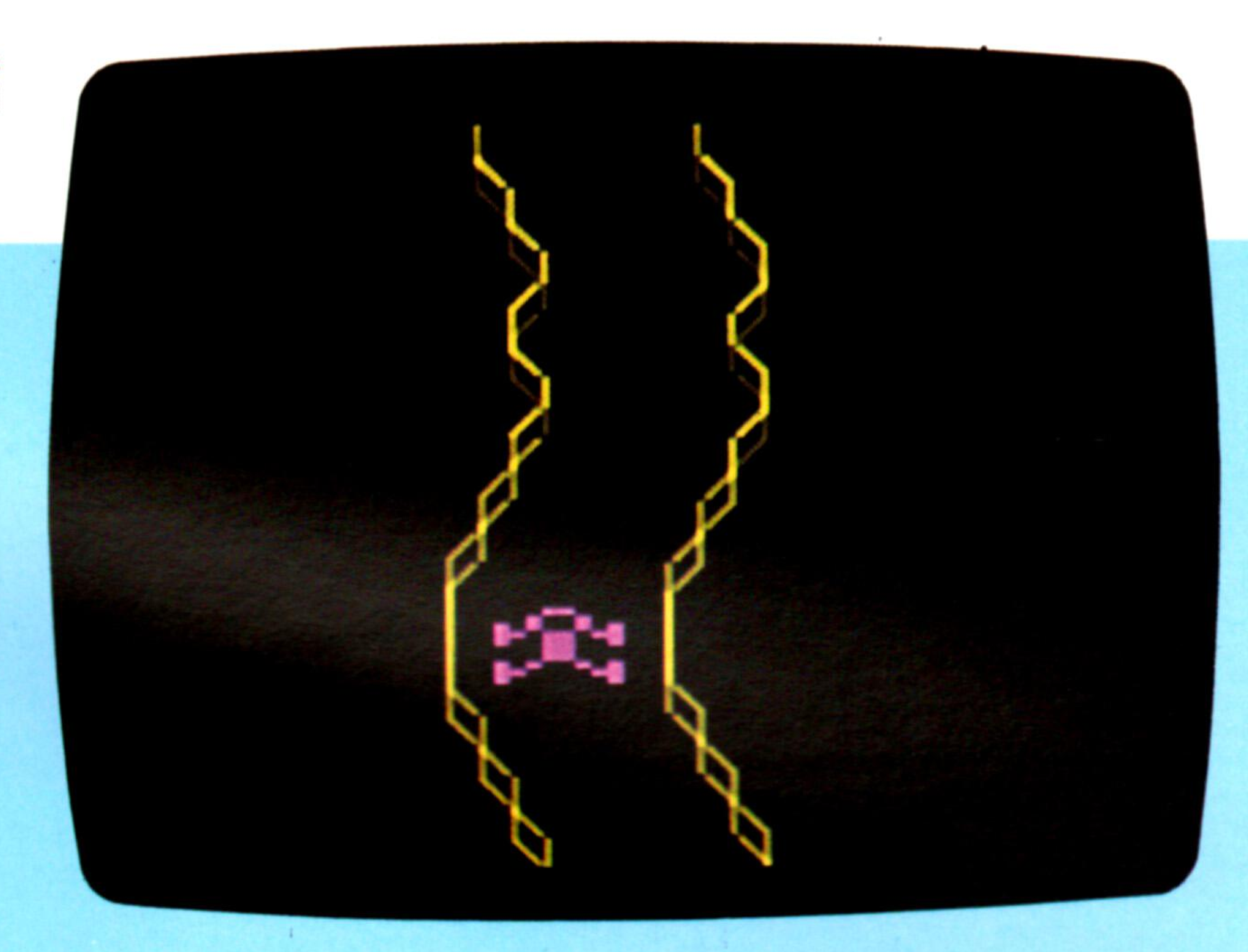
BBC Micro:
Introducir las siguientes modificaciones:

```
6820 CLS
6895 I$=GET$
6910 CLS
6970 I$=GET$
7010 CLS
7060 CLS
7155 I$=GET$
```

Gran Premio 2

He aquí la segunda parte de un juego del cual ya hemos proporcionado un programa para su ordenador Atari. No olvide que la palanca de mando es obligatoria

Para conducir su bólido utilice la palanca de mando. Si logra llegar a la meta, será bonificado: el valor de la distancia recorrida se duplicará.



```
10 REM ** GRAN PREMIO 2 – P.BUNN **
15 PI=5:GOSUB 3000:ACCIDENTE=1000
20 X2=14:X=116:S=PEEK(106)–8:Q=S*256:FOR
   N=Q+512 TO Q+640:POKE N,0:NEXT N:POKE
   54279,S
30 POKE 559,46:POKE 53248,X:POKE 704,216:POKE
   704,70:POKE 53256,1
40 FOR N=Q+552 TO Q+561:READ A:POKE N,A:NEXT N
50 DATA 129,195,165,24,24,153,219,165,36,24
60 GRAPHICS 0:SETCOLOR 2,0,0:POKE 53277,3:POKE
   559,46
65 POKE 752,1:POKE 53278,A
66 FOR P=0 TO 23:POKE 201,X2:? ,S2$:NEXT P
70 S=STICK(0):X=X+(S=7)*2:X=X–(S=11)*2:POKE
   53248,X
72 SOUND 0,40,10,15:SC=SC+PI:SOUND 0,0,0,0
75 IF O THEN O=0:Z=2:GOTO 110
80 A=PEEK(53770)
90 IF A>85 AND A<170 THEN Z=2
100 IF A>170 AND X2<15 THEN Z=3:O=1
105 IF A<85 AND X2>2 THEN X2=X2–1:Z=1:O=1
110 POKE 201,X2
115 IF A=192 AND NOT I THEN ? ,F$:I=1:GOTO 145
120 IF Z=1 THEN ? ,S1$
130 IF Z=2 THEN ? ,S2$
140 IF Z=3 THEN ? ,S3$:X2=X2+1
145 Y=PEEK(53252):IF Y<>0 AND I THEN GOTO 2000
150 IF Y<>0 THEN GOTO ACCIDENTE
160 GOTO 70
1000 REM ** ACCIDENTE!!!! **
1010 N=INT(RND(0)*10):POKE
     Q+552+N,PEEK(53770):SOUND
     0,RND(0)*20+20,80,15:POKE 704,PEEK(53770)
1020 IF PEEK(53770)<240 THEN 1010
1030 POKE 53248,0:? :? "SUS PUNTOS:",SC :? :? :? "PULSE
     UNA TECLA.":POKE 764,255
1035 SOUND 0,0,0,0
1040 IF PEEK(764)=255 THEN 1040
1050 RUN
2000 FOR Y=0 TO 255:POKE 710,Y:NEXT Y
2005 ? CHR$(125);"        PUNTOS:";SC:POKE 710,0
2010 ?"       BONO    :1000":B=1000
2020 FOR G=1 TO 1000 STEP
     10:B=B–10:SC=SC+10:POSITION 14,0:? SC:POSITION
     14,1:? B;"     ":SOUND 0,G/4,10,10:NEXT G
2030 SOUND 0,0,0,0
2040 ?  :PI=PI*2:? "1 KM VALE    ";PI;" PUNTOS"
2045 RESTORE :I=0
2050 GOTO 20
3000 DIM F$(8),S1$(8),S2$(8),S3$(8)
3010 RESTORE 3000:FOR P=1 TO 8
3020 READ A,B,C,D
3030 F$(P,P)=CHR$(A)
3040 S1$(P,P)=CHR$(B)
3050 S2$(P,P)=CHR$(C)
3060 S3$(P,P)=CHR$(D)
3070 NEXT P
3080 RESTORE :RETURN
3090 DATA 160,6,22,7,198,32,32,32
3100 DATA 201,32,32,32,206,32,32,32
3110 DATA 201,32,32,32,211,32,32,32
3120 DATA 200,32,2,32,160,6,32,7
```

Para el Spectrum

Ahora ofrecemos para el Spectrum un programa que permite controlar el brazo-robot a través de la interface construida anteriormente

El software requerido para controlar el brazo-robot debe basarse en código máquina, porque los servomotores digitales empleados requieren impulsos regulares cada quincuagésima de segundo. Tales velocidades se hallan fuera del alcance de un programa en BASIC. Al igual que las versiones para el Commodore y el BBC Micro, este código máquina se ejecuta sobre una base de interrupciones, pero difiere significativamente porque el Spectrum utiliza un procesador Z80 en vez de los de la serie 65XX que emplean las otras dos máquinas. La mejor forma de tratar interrupciones en código máquina de Z80 es utilizar interrupciones en modo IM 2. Para explicarlo en términos simples, cuando se establece esta modalidad de interrupciones y se genera una interrupción para actualizar la visualización en pantalla (afortunadamente, alrededor de cada quincuagésima de segundo), el procesador toma la dirección de inicio del código de servicio de interrupción de dos fuentes. El byte *hi* de la dirección está retenido en el registro I y el byte *lo* se toma del estado del bus de datos cuando se produjo la interrupción.

El sistema está diseñado para comunicaciones con periféricos conectados al bus de datos que puedan controlar el valor retenido en el bus. Sin embargo, en nuestra aplicación no conocemos el estado del bus de datos y, por lo tanto, hemos de acomodar las 256 posibilidades. El programa, por consiguiente, prepara una página de memoria para retener la dirección de comienzo real de nuestro programa. Llenar una página con #FBs y apuntar a la página mediante el establecimiento del registro I provoca una interrupción IM 2 que empezará a ejecutar código en la posición #FBFB. (Esta técnica se describirá en profundidad en una futura serie de código máquina dedicada al OS del Spectrum.)

El resto del código máquina utiliza los mismos principios de operación que en las versiones 65XX, produciendo una corriente de impulsos hacia la puerta E/S 31, la puerta asociada a nuestra interface. El programa de control en BASIC, asimismo, sigue líneas similares a las desarrolladas para el BBC Micro y el Commodore 64, permitiendo la programación de secuencias desde el teclado y su ulterior reproducción.

Programador de secuencias para el brazo en el Spectrum

Cargador en BASIC

```
  10 REM ********************************************
  15 REM ********************************************
  17 REM **                                        **
  20 REM **            controlador                 **
  25 REM **          brazo spectrum                **
  30 REM **                                        **
  40 REM ********************************************
  50 REM ********************************************
  60 :
  70 CLEAR 30000
  80 LET sa=64017:REM comienzo c/m cuña
  82 LET np=64008:REM direccion comienzo nuevapos
  84 LET dl=64016:REM direccion factor demora
  86 LET en=64148:REM cuña apagada
  88 LET sm=64151:REM dir comienzo movimiento uniforme
  90 LOAD "ARM.HEX"CODE sa
 100 GO SUB 1000:REM preparacion
 110 CLS
 120 PRINT AT 6,2;"te gustaria:"
 130 PRINT AT 9,2;"1..programar nueva secuencia"
 140 PRINT AT 10,2;"2..añadir movimientos a un programa"
 150 PRINT AT 11,2;"3..cargar un archivo"
 160 PRINT AT 12,2;"4..salir del programa"
 170 LET g$=INKEY$:IF g$="" THEN GO TO 170
 180 IF g$="1" THEN LET c=1:LET lm=0
 190 IF g$="1" OR g$="2" THEN GO SUB 2000:GO SUB 3000:GO SUB
     4000:GO SUB 5000
 200 IF g$="3" THEN GO SUB 6000
 210 IF g$="4" THEN CLS:RANDOMIZE USR en:STOP
 220 GO TO 110
 230 :
1000 REM **** preparacion ****
1010 LET c=1:LET ns=4:REM n. de servos
1020 LET lm=0:LET oc=0:LET df=10:REM factor de demora
1030 LET mc=100:REM max. contador
1040 DIM r(mc,ns):REM matriz posiciones teclas
1050 DIM l(1):REM matriz datos ultimomax
1060 FOR i=0 TO 3:POKE np+i,0:NEXT i
1070 RANDOMIZE USR sa
1090 RETURN
1099 :
2000 REM **** informe ****
2010 CLS
2020 PRINT AT 3,2;"por favor utiliza"
2030 PRINT AT 4,2;"n/m...para i/d"
2040 PRINT AT 5,2;"p/l...para 1.er brazo ar/ab"
2050 PRINT AT 6,2;"a/z...para 2.º brazo ar/ab"
2060 PRINT AT 7,2;"x/c...para pinza abrir/cerrar"
2070 PRINT AT 8,2;"s.....guardar una posicion"
2080 PRINT AT 9,2;"q.....retornar al menu"
2090 PRINT AT 10,2;"r....mover a posicion guardada"
2100 PRINT AT 11,2;"v/b...siguiente y anterior contador"
2110 PRINT AT 12,2;"e.....establecer nuevo contador"
2120 PRINT AT 13,2;"i/d...para aum/dism velocidad"
2130 PRINT AT 1,2;"contador=";c;"  ";
2135 FOR i=1 TO 4:PRINT r(c,i);"  ";:NEXT  i:PRINT
2140 RETURN
2190 :
3000 REM **** programar brazo ****
3010 POKE dl,1
3020 LET dx=5
3030 LET a$=INKБY$: IF a$="" THEN GO TO 3030
3035 IF a$="n" THEN LET p=PEEK(np)+dx:IF p<256 THEN POKE np,p
3040 IF a$="m" THEN LET p=PEEK (np)-dx:IF p>0 THEN POKE np,p
3050 IF a$="p" THEN LET p=PEEK (np+1)+dx: IF p<256 THEN POKE
     np+1,p
3060 IF a$="l" THEN LET p=PEEK(np+1)-dx:IF p>0 THEN POKE np+1,p
3065 IF a$="a" THEN LET p=PEEK (np+2)+dx:IF p<256 THEN POKE
     np+2,p
3070 IF a$="z" THEN LET p=PEEK (np+2)-dx:IF p>0 THEN POKE
     np+2,p
3080 IF a$="x" THEN LET p=PEEK (np+3)+3*dx:IF p<256 THEN POKE
     np+3,p
3090 IF a$="c" THEN LET p=PEEK (np+3)-3*dx:IF p>0 THEN POKE
     np+3,p
3100 IF a$="r" THEN FOR i=1 TO 4:POKE np+i-1,r(c,i):NEXT i
3110 IF a$="v" THEN LET c=c+1:IF c>mc THEN LET c=1
3120 IF a$="b" THEN LET c=c-1:IF c<1 THEN LET c=mc
3130 IF a$="e" THEN PRINT AT 2,20;"valor contador":INPUT c
3140 IF A$="s" THEN FOR i=1 TO 4:LET r(c,i)=PEEK (np+i-1):NEXT i:
     LET c=c+1
3145 IF a$="i" THEN LET dx=dx+1
3147 IF a$="d" THEN LET dx=dx-1:IF dx<1 THEN LET dx=1
3150 IF c>lm THEN LET lm=c
3160 IF oc<>c AND c>1 THEN PRINT AT 1,2;"contador=";c-1;"  ";:FOR
     i=1 TO 4:PRINT r(c-1,i);"  ";: NEXT i:PRINT
3170 LET oc=c
3190 IF a$<>"q" THEN GO TO 3030
3200 RETURN
3900 :
4000 REM **** reproducir secuencia ****
4010 CLS
4020 PRINT AT 12,2;"reproducir secuencia s/n, r repite"
4030 LET a$="INKEY$:IF a$<>"s" AND a$<>"n" AND a$<>"r" THEN
     GO TO 4030
4040 IF a$="n" THEN RETURN
4050 IF a$="r" THEN PRINT AT 14,1;"factor de demora 1-255";:INPUT df
4060 IF df<1 OR df>255 THEN GO TO 4050
4070 POKE dl,df:REM establecer registro demora
4080 FOR i=l TO lm
4090 PRINT AT 1,2;"n.en secuencia=";i;"  ";
4100 FOR s=1 TO 4
4110 POKE np+s-1,r(i,s)
```



```
4120 NEXT s:NEXT i
4140 GO TO 4010
4990 :
5000 REM **** guardar un archivo ****
5010 CLS
5020 PRINT AT 12,2;"guardar secuencia (s/n)?"
5030 LET g$=INKEY$:IF g$<>"s" AND g$<>"n" THEN GO TO 5030
5040 IF g$="n" THEN RETURN
5050 LET l (1)=lm
5060 INPUT "nombre archivo";f$
5065 PRINT "pulsar play y record"
5067 RANDOMIZE USR en:REM cuña apagada
5070 SAVE f$+".lm" DATA l()
5080 SAVE f$+".r" DATA r()
5085 RANDOMIZE USR sa:REM cuña encendida otra vez
5090 RETURN
5900 :
6000 REM **** cargar un archivo ****
6010 CLS
6020 PRINT AT 12,2;"cargar un archivo (s/n)?"
6030 LET g$=INKEY$:IF g$<>"s" AND g$<>"n" THEN GO TO 6030
6040 IF g$="n"THEN RETURN
6045 INPUT "nombre archivo";f$
6050 FOR i=1 TO mc
6060 FOR j=1 TO ns
6070 LET r(i,j)=0
6080 NEXT j:NEXT i
6090 RANDOMIZE USR en:REM cuña apagada
6100 LOAD f$+".lm" DATA l()
6110 LET lm=l(1):LET c=lm:LET oc=0
6120 LOAD f$+".r" DATA r()
6125 RANDOMIZE USR sa:REM cuña encendida otra vez
6130 RETURN
```

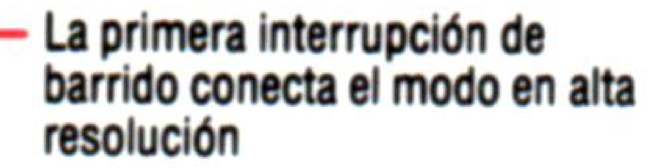
Listado assembly

```
                1000 ;CONTROLADOR BRAZO SPECTRUM
                1010
001F            1020 PORT:     EQU   31
F900            1030           ORG   # F900
F900            1040 MOTTAB    DEFS  256
FA00            1050 ANGTAB    DEFS  8
FA08            1060 NEWPOS    DEFS  8
FA10            1070 DELAY:    DEFS  1
                1080
                1090 ;PREPARAR TABLA VECTOR
FA11 2100FC     1100 INIT:     LD    HL,#FC00
FA14 01FB00     1110           LD    BC,#00FB
FA17 71         1120 LOOP1:    LD    (HL),C
FA18 23         1130           INC   HL
FA19 10FC       1140           DJNZ  LOOP1
FA1B 71         1150           LD    (HL),C
FA1C 3EFC       1170           LD    A,#FC
FA1E ED47       1180           LD    I,A
                1190 ;INIT TABLAS A FF
                1200 ;
FA20 2100F9     1210           LD    HL,MOTTAB
FA23 3EFF       1220           LD    A,#FF
FA25 06FF       1230           LD    B,#FF
FA27 77         1240 LOOP2:    LD    (HL),A
FA28 2C         1250           INC   L
FA29 10FC       1260           DJNZ  LOOP2
FA2B 2100FA     1270           LD    HL,ANGTAB
FA2E 0610       1280           LD    B,16
FA30 77         1290 LOOP2A    LD    (HL),A
FA31 2C         1300           INC   L
FA32 10FC       1305           DJNZ  LOOP2A
FA34 ED5E       1310           IM    2
FA36 C9         1320           RET
                1380 ; MANEJADOR INTERRUPCIONES
                1390
FA37 F3         1410 HANDLE    DI
FA38 F5         1420           PUSH  AF
FA39 C5         1430           PUSH  BC
FA3A D5         1440           PUSH  DE
FA3B E5         1450           PUSH  HL
FA3C FF         1460           RST   #38;RUTINA NML
FA3D F3         1470           DI
FA3E CD47FA     1480           CALL  EVENT ;NUESTRA RUTINA
FA41 E1         1490           POP   HL
FA42 D1         1500           POP   DE
FA43 C1         1510           POP   BC
FA44 F1         1520           POP   AF
FA45 FB         1530           EI
FA46 C9         1540           RET
                1550
                1560 ;
                1570 ;++++ RUTINA EVENT ++++
                1580 ;
FA47 0608       1590 EVENT:    LD    B,8
FA49 3E7F       1600           LD    A,#7F
FA4B 16F0       1610           LD    D,#F0
FA4D 2158FA     1620           LD    HL,FIX+2
FA50 DD2100FA   1630           LD    IX,ANGTAB
FA54 3607       1635           LD    (HL),7
FA56 DD5E07     1640 FIX:      LD    E,(IX+7)
FA59 12         1650           LD    (DE),A
FA5A 0F         1660           RRCA
FA5B 35         1670           DEC   (HL)
FA5C 10F8       1680           DJNZ  FIX
                1690 ;
                1740 ;PREP MOTTAB PARA PORTSEND
                1750 ;
FA5E 0600       1760           LD    B,0
FA60 3EFF       1770           LD    A,#FF
FA62 2100F9     1780           LD    HL,MOTTAB
FA65 A6         1790 LOOP3:    AND   (HL)
FA66 77         1800           LD    (HL),A
FA67 2C         1810           INC   L
FA68 10FB       1820           DJNZ  LOOP3
                1830 ;
                1840 ;++++ COMENZAR IMPULSOS ++++
                1850 ;
FA6A 3EFF       1860           LD    A,#FF
FA6C D31F       1870           OUT   (PORT),A
                1880 ;
                1885 ;++++ LLAMAR DEMORA ++++
FA6E D5         1887           PUSH  DE
FA6F 1E04       1889           LD    E,4
FA71 16FF       1890           LD    D,#FF
FA73 CD8BFA     1892           CALL  OLOOP
FA76 D1         1894           POP   DE
                1920 ;
                1930 ;++++ ENVIAR MOTTAB A PORT ++++
                1940 ;
FA77 0E1F       1950           LD    C,PORT
FA79 06FF       1960           LD    B,#FF
FA7B 2E00       1970           LD    L,00
FA7D EDB3       1980           OTIR
                1990 ;
                2000 ;++++ RESTAURAR MOTTAB A FF ++++
+
FA7F 3EFF       2010           LD    A,#FF
FA81 0600       2020           LD    B,0
FA83 2100F9     2030           LD    HL,MOTTAB
FA86 77         2040 LOOP4:    LD    (HL),A
FA87 2C         2050           INC   L
FA88 10FC       2060           DJNZ  LOOP4
FA8A C9         2070           RET
                2080 ;++++ BUCLE DEMORA ++++
                2090 ;
FA8B 1D         2100 OLOOP:    DEC   E
FA8C C8         2110           RET   Z
FA8D 15         2120 ILOOP:    DEC   D
FA8E CA8BFA     2130           JP    Z, OLOOP
FA91 C38DFA     2140           JP    ILOOP
                2200 ;
                2210 ;++++ RESTAURAR IM 1 ++++
                2220
FA94 ED56       2230 REST:     IM    1
FA96 C9         2240           RET
                2250 ;
                2340 ;++++ MOVEDOR UNIFORME ++++
                2350 ;
FA97 0E04       2360 START:    LD    C,4
FA99 0604       2370           LD    B,4      ;PREPARAR
                                     CONTADORES
FA9B 2130FA     2380           LD    HL,ANGTAB+3
FA9E 110BFA     2390           LD    DE,NEWPOS+3
FAA1 7E         2400 NEXMOT    LD    A,(HL)
FAA2 EB         2410           EX    DE,HL
FAA3 BE         2420           CP    (HL)
FAA4 EB         2430           EX    DE,HL
FAA5 280A       2440           JR    Z,MOVED
FAA7 3804       2450           JR    C,ADD
FAA9 35         2460           DEC   (HL)
FAAA C3B2FA     2470           JP    NEXT
FAAD 34         2480 ADD:      INC   (HL)
FAAE C3B2FA     2490           JP    NEXT
FAB1 0D         2500 MOVED:    DEC   C
FAB2 2B         2510 NEXT:     DEC   HL
FAB3 1B         2520           DEC   DE
FAB4 10EB       2530           DJNZ  NEXMOT
FAB6 F5         2540           PUSH  AF
FAB7 D5         2550           PUSH  DE
FAB8 ED5B10FA   2560           LD    DE,(DELAY)
FABC 16FF       2570           LD    D,#FF
FABE CD8BFA     2580           CALL  OLOOP ;LLAMAR DEMORA
FAC1 D1         2590           POP   DE
FAC2 F1         2600           POP   AF
FAC3 20D2       2610           JR    NZ,START
FAC5 C9         2620           RET
                3000 ;++++ PREPARAR MANEJADOR ++++
                3010 ;++++ DIRECCION DE SALTO ++++
                3020 ;
FBFB            3030           ORG   #FBFB
FBFB F3         3040           DI
FBFC C337FA     3050           JP    HANDLE
Pass 2 errors:  00

ADD      FAAD    ANGTAB  FA00
DELAY    FA10    EVENT   FA47
FIX      FA56    HANDLE  FA37
ILOOP    FA8D    INIT    FA11
LOOP1    FA17    LOOP2   FA27
LOOP2A   FA30    LOOP3   FA65
LOOP4    FA86    MOTTAB  F900
MOVED    FAB1    NEWPOS  FA08
NEXMOT   FAA1    NEXT    FAB2
OLOOP    FA8B    PORT    001F
REST     FA94    START   FA97
```

Imágenes a trozos

Vamos a analizar la técnica de producir al mismo tiempo dibujo y texto, tan frecuente en programas de juegos


La primera interrupción de barrido conecta el modo en alta resolución


La segunda interrupción de barrido devuelve al modo de texto

Dimension Graphics

Las dos caras de la historia
Muchos juegos de aventuras comerciales escritos para el Commodore 64 emplean técnicas de interrupción de barrido para obtener dibujos en alta resolución y texto simultáneamente. *Spiderman*, cuyo autor es Scott Adams y que ilustramos aquí, utiliza el tercio superior de la pantalla para visualizar una posición en el juego; los restantes dos tercios se usan para describir la escena en el modo normal de visualización de textos. El explorador de barrido de la TV está programado para generar dos interrupciones cada vez que se inspecciona la pantalla: la primera en la parte superior y la segunda aproximadamente un tercio más abajo. Cada vez que sucede una interrupción de barrido se llama una rutina de interrupción, lo que hace que la visualización bascule entre los modos de alta resolución y el de texto

Para visualizar los datos de video, el chip VIC-II debe leerlos de la memoria. Esto se consigue haciendo que el VIC-II acceda a algunas líneas (no todas) del bus de direcciones, y a todas las líneas del bus de datos. El proceso por el cual el VIC-II lee la ROM y la RAM compartida con el 6510 se denomina *acceso directo a memoria* (DMA). El VIC-II debe, en teoría, usar las líneas de dirección o de datos en los ratos en que el 6510 no las utilice, en cuyo caso la actuación de ambos procesadores es "transparente" recíprocamente. Pero el 6510 es un procesador bastante modesto con muy pocos registros internos y ninguna de las instrucciones del 6510 consume más de siete ciclos de reloj (algunas tan sólo tres o cuatro).

Ya que el VIC-II debe leer todo un conjunto de datos, sobre todo cuando visualiza varios sprites, no hay tiempo material para que pueda actuar de la forma transparente antedicha. Por ello, el VIC-II tiene una línea de control especial, llamada de *bus disponible* (BA: *bus available*) que puede emplear cuando desee enviar una señal de "precedencia" al 6510. Lo cual le permite reservarse tanto tiempo como necesite para leer los datos de video.

Cuando la línea BA se pone baja (*low*) quiere decir que el chip del video solicita tiempo al 6510. Entonces el otro chip dispone del tiempo suficiente para rematar la instrucción que esté realizando antes de que el VIC-II baje la línea de *control de habilitación de direcciones* (AEC) y desactive sin más los manejadores del bus de direcciones del 6510. Esto equivale a desarmar por sorpresa al 6510 con otra arma más poderosa: éste no se da cuenta de que ha sido puesto fuera de combate.

El tiempo robado del 6510 por el chip VIC-II es importante. Por ejemplo, las direcciones del puntero de caracteres deben ser tomadas después de cada octava línea de barrido visualizada en la pantalla (ya que los caracteres tienen ocho filas de pixels) y cada línea necesita 40 accesos consecutivos para tomar los punteros de la memoria de video (hay 40 caracteres por fila de pantalla). En el sistema PAL, el chip del video renueva una línea sí y otra no de las 625 líneas de barrido en la pantalla del televisor aproximadamente 25 veces por segundo; el sistema americano NSTC tiene 524 líneas, y las renueva 30 veces por segundo. Esto suma una buena cantidad de tiempo. De hecho, una sencilla operación aritmética muestra que el efecto neto es el freno en la velocidad del 6510 en el 15 o 20 % aproximadamente.

Los DMA no tienen ningún efecto notable sobre el funcionamiento interno de la máquina. Pero cuando está en cuestión el tiempo real de E/S estas "ausencias" imprevistas del 6510 pueden convertirse en un problema, por ejemplo, en el funcionamiento de la cassette. En tales casos puede ser necesario desconectar la pantalla (por medio de POKE 53265,11) durante la E/S, y volverla a conectar (con POKE 53265,27) después de finalizada la E/S. El que el empleo del acceso directo a la memoria que hace el VIC-II resulte problemático durante una E/S dependerá del dispositivo externo y del método empleado para comunicarse con él. En muchos casos, como el acceso a un disco, no será necesario desconectar la pantalla.

Partir la pantalla

Una característica interesante del Commodore 64 consiste en que, si se programa hábilmente, es posible dividir la pantalla visual en dos modos de resolución alta y baja. Esto puede hacer, por ejemplo, que un gráfico se esté visualizando al mismo tiempo que se imprime un texto o se realiza cualquiera otra interacción con el usuario en baja resolución sólo en una porción de la pantalla.

El programa para partir la pantalla que proporcionamos aquí muestra varios de los aspectos tratados. Se ha preparado una pequeña pantalla en alta resolución detrás de la ROM del intérprete de BASIC y se visualiza continuamente en el tercio superior de la pantalla. Al mismo tiempo, los dos tercios inferiores quedan en el modo normal de baja resolución.

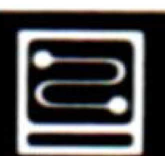
Ya hemos estudiado el empleo de los vectores de la RAM en el Commodore 64. Mostramos cómo el cambio del valor de un puntero de dos bytes nos permitía desviar una rutina del sistema operativo a un código creado por nosotros. Para obtener una partición de la pantalla desviaremos la rutina IRQ de una manera algo sibilina.

Hay varios modos de disparar una IRQ en el 6510. La manera habitual es que uno de los temporizadores del CIA ponga un bit del registro del flag de interrupción (IFR) en la dirección 53273 ($D019) cada sesentavo de segundo. Esto provoca la ejecución de la rutina de servicio IRQ, la cual, entre otras cosas, inspecciona la matriz de conexiones del teclado para saber si se pulsó alguna tecla. También se ponen a uno los bits del IFR por medio del chip VIC-II cuando se cumplen ciertos eventos, tales como el caso de colisión de dos sprites o cuando el contador de barrido alcance un valor preestablecido. Hay otro registro, el *registro activador de interrupciones* (IER) en la dirección 53274 ($D01A) que actúa como un conector activador de la línea IRQ del 6510.

Si los bits correspondientes del IER son puestos por el programador a 1, entonces los bits activados del IFR dispararán una IRQ en el 6510.

La idea general del programa es la siguiente:

1) Cuando sucede una interrupción de barrido en la parte superior de la pantalla, activar el modo de mapa de bits y después establecer la interrupción



de barrido a mitad de la pantalla. Retorno de la interrupción (RTI).

2) A la interrupción en la mitad de la pantalla, volver a establecer el modo en baja resolución y establecer después la siguiente interrupción de barrido para que suceda en la parte superior de la pantalla (seguido de otro RTI).

Obsérvese un aspecto algo espinoso de esta idea: parte de la memoria normal de pantalla correspondiente al tercio superior de la pantalla se ha empleado para información del color en la pantalla de alta resolución: los dos tercios restantes se emplean para datos de caracteres de la manera habitual.

Sin embargo, el programa tiene una dificultad. Las interrupciones de barrido sucederán cada cincuentavo de segundo y deben atenderse inmediatamente, pues de lo contrario el barrido tendrá que seguir desplazándose, y visualizará datos erróneos. Ahora bien, en circunstancias normales, la rutina de servicio de IRQ se dispara cada sesentavo de segundo. La rutina de servicio es bastante larga y si ocurre justamente antes de la interrupción de barrido, se producirá un retraso antes de llegar a nuestra cuña de barrido del programa. Esto hay que evitarlo a toda costa, pues si no es así la pantalla basculará en la frontera de alta y la baja resolución.

La solución a este problema consiste en obligar a la rutina IRQ a que se ejecute inmediatamente después del código de la cuña de barrido. Esto se consigue desactivando el temporizador habitual del sesentavo de segundo, y así se desactiva el disparador habitual de IRQ, saltando a la rutina IRQ justamente después de haber ejecutado la cuña de barrido. Esto obliga a que los dos fragmentos de código se ejecuten en sincronía el uno con el otro. Naturalmente esto sólo significa que el teclado será marginalmente inspeccionado con menor frecuencia, lo que tiene el útil efecto lateral de acelerar algo la ejecución del BASIC, sin mayores daños.

Finalmente, hemos puesto la pantalla en alta resolución detrás de la ROM del intérprete de BASIC. Esto hace que el "puntearla" en BASIC resulte algo dificultoso, pues el lenguaje no puede leer el área de RAM que está detrás de la ROM del BASIC, y no podemos realizar los necesarios AND y OR para establecer los pixels individuales. No obstante, un POKE puede pasar por la ROM del BASIC y llegar hasta la RAM trasera, permitiendo el *plotting* simple desde el BASIC. En los programas de sólo lenguaje máquina no necesitamos la ROM del intérprete, y como no representan más que 8 K de espacio desperdiciado, se desconecta. Unos pequeños arreglos en el listado fuente le permitirán reposicionar la pantalla en alta resolución de modo que pueda usarse con mayor facilidad desde el BASIC.

rograma de partición de pantalla para C64

rgador del BASIC

```
REM ************************************
REM **  CARGADOR BASIC DEL PROGRAMA     **
REM **      PARTICION PANTALLA 64       **
REM ** NOTA — EN ESTA VERSION           **
REM **  LA ALT. RES. ESTA BAJO EL BASIC **
REM **   LUEGO NO SE PUEDE USAR PEEK    **
REM **          EN SUBR. PLOT           **
REM ************************************
:
REM CARGA Y PRUEBA DE PARTICION PANTALLA **
DATA76,50,192,49,234,173,25,208
DATA240,37,169,255,160,21,141,25
DATA208,173,0,221,73,2,141,0,221
DATA173,17,208,73,32,141,17,208,41
DATA32,240,4,169,121,160,255,141
DATA18,208,140,24,208,108,3,192
DATA173,14,220,41,254,141,14,220
DATA173,20,3,141,3,192,173,21,3
DATA141,4,192,169,5,141,20,3,169
DATA192,141,21,3,173,17,208,41,95
DATA141,17,208,169,0,141,18,208
DATA169,255,141,25,208,169,1,141
DATA26,208,165,1,41,254,133,1,169
DATA0,133,247,169,160,133,248,160
DATA0,162,28,169,0,145,247,200,208
DATA251,230,248,202,208,246,169
DATA188,133,248,160,0,162,4,169
DATA183,145,247,200,208,251,230
DATA248,202,208,246,165,1,9,1,133
DATA1,96,255
DATA20633:REM*CHECKSUM*
CC=0:FOR I=0TO160
READX:POKE49152+I,X:REM INSERTA CODIGO
CC=CC+X:NEXT
READX:IFX<>CCTHENPRINT"CHECKSUM ERROR":STOP
REM ** PRUEBA PARTICION PANTALLA **
SYS49152:REM PARTICION PANTALLA
CM=40960 : REM INICIO ALT. RES.
REM ** EJE VERT TRAZO **
X=160 : FOR Y=0 TO 70
GOSUB 1400 : REM CALCULA DIRECCION
POKE M+R,2 ↑ (7-C): REM TRAZA PUNTO
NEXT
REM ** EJE HORIZ TRAZO **
Y=37 : FOR X=0 TO 319
GOSUB 1400 : REM CALCULA DIRECCION
POKE M+R,255 : REM TRAZA SEGMENTO
NEXT
END
REM ** S/R CALCULO **
1410 U=INT(Y/8):V=INT(X/8)
1420 R=YAND7:C=XAND7
1430 M=CM + (40*U+V)*8
1440 RETURN
```

Listado assembly

```
        ; ************************************************
        ; ************************************************
        ; **              PARTICION PANTALLA 64         **
        ; ************************************************
        ; ************************************************
        ;
        ;
        *= $C000
        JMP START
        ;
        MEM1  = $F7
        MEM2 = $F9
        MEM3 = $FB
        SCN = $FD
        CINV = $314         ; VECTOR IRQ
        TEMP IRQ *=*+2
        ;
NEWIRQ =*                   ; NUEVO INICIO CUÑA IRQ
        ;
        LDA   $D019         ; MIRA EL ESTADO DE INTERRUPCION
        BEQ   NOTVIC        ; VINO LA IRQ DEL VIC?
        LDA   #$FF
        LDY   #$15
        STA   $D019         ; BORRA LATCH INT.
        LDA   $DD00
        EOR   #$02
        STA   $DD00         ; CAMBIA BANCOS
        LDA   $D011
        EOR   #$20          ; CAMBIA BIT DEL MODO MAPA DE BITS
        STA   $D011
        AND   #$20          ; COMPARA EL BIT BMM
        BEQ   LSCN          ; PARTE INFERIOR PANTALLA
        LDA   #$79          ; PONE EL BARRIDO EN LINEA 22
        LDY   #$FF
LSCN
        STA   $D012         ; ESTABLECE COMP. NUEVO BARRIDO
        STY   $D018
NOTVIC
        JMP   (TEMP IRQ)
        ;
START =*                    ; INICIALIZA CUÑA IRQ
        ;
        LDA   $DC0E
        AND   #$FE          ; DESACTIVA TEMPORIZADOR RASTREO
        STA   $DC0E         ; CIA #1
        LDA   CINV
        STA   TEMP IRQ      ;
        LDA   CINV+1        ; CAMBIA VECTOR IRQ
        STA   TEMP IRQ+1    ;
        LDA   #<NEWIRQ
        STA   CINV
        LDA   #>NEWIRQ
        STA   CINV+1
        LDA   $D011
        AND   #$5F          ; RESTABLECE MODO BMM
        STA   $D011
        LDA   #$00
        STA   $D012
        LDA   #$FF
        STA   $D019         ; RESTABLECE LATCHES DE IRQ
        LDA   #$01
        STA   $D01A         ; ACTIVA COMPARACION BARRIDO
        LDA   $01
        AND   #$FE
        STA   $01           ; DESCONECTA LA ROM DEL BASIC
        LDA   #$00
        STA   MEM1          ; EST. PUNTS. P. 0 PARA APUNTAR AL
        LDA   #$A0          ; INICIO AREA ALT. RES. DETRAS DE LA
        STA   MEM1+1        ; ROM DEL BASIC
        LDY   #$00
        LDX   #$1C
        LDA   #$00          ; BORRA ALT. RES.
DDD
        STA   (MEM1),Y
        INY
        BNE   DDD
        INC   MEM1+1
        DEX
        BNE   DDD
        LDA   #$BC
        STA   MEM1+1
        LDY   #$00
        LDX   #$04
        LDA   #$B7          ; ESTABLECE COLOR
DDE
        STA   (MEM1),Y
        INY
        BNE   DDE
        INC   MEM1+1
        DEX
        BNE   DDE
        LDA   $01
        DRA   #$01
        STA   $01           ; RESTAURA ROM BASIC
        RTS                 ; VUELTA AL BASIC
```

Héroe temerario

"Impossible mission" (Misión imposible), de Epyx, es un juego con excelentes gráficos y un guión con "carrera contra el tiempo" incluida

Impossible mission: Para el Commodore 64
Editado por: CBS (Columbia Broadcasting System) Software, 3-5 Rathbone Place, London W1, Gran Bretaña
Autor: Dennis Caswell
Palanca de mando: Necesaria
Formato: Disco o cassette

Entre los diversos tipos de programas de juegos, uno de los más populares es el juego de plataformas. En el mismo, el jugador (en el papel de héroe) debe obtener objetos que están situados en plataformas dentro de las diversas pantallas. Por lo general el héroe penetra en la pantalla por una de las esquinas inferiores y debe alcanzar las plataformas mediante una serie de ascensores, escaleras y trampolines (o lo que sea que haya puesto el programador a su disposición). A menudo, objetos vitales tales como llaves o un importante tesoro están colocados en una posición particularmente difícil que implica un considerable esfuerzo para determinar la mejor forma de llegar hasta ellos.

Los juegos de este tipo se complican aún más mediante la presencia de alienígenas u otras figuras gráficas peligrosas que se han de evitar, porque el mero hecho de tocarlas suele tener consecuencias fatales. Estos gráficos pueden ser estáticos o bien desplazarse siguiendo patrones preestablecidos. Además, muchos de estos sprites poseen capacidad de "disparo". Según cuáles sean las intenciones del programador, los movimientos de estos enemigos pueden ser más o menos inteligentes. Muchos de ellos se moverán hacia atrás o hacia adelante a lo largo de un único camino, mientras que otros pueden estar programados para "apuntar" hacia el héroe, limitando su libertad de acción.

Los juegos de plataformas le ofrecen varias ventajas al programador. Entre ellas la más importante es el escaso código necesario para la implementación del juego. Una vez que el programador ha definido el diseño gráfico del héroe, los guardias, el tesoro, las plataformas, las escaleras y los ascensores, todas las diferentes "habitaciones" se pueden programar con sólo reposicionar los diversos elementos. Es obvio que ello supone un gran ahorro de espacio, que se puede utilizar para mejorar el nivel de detalle del juego o incrementar el número de habitaciones.

Impossible mission saca el máximo partido de la capacidad de los juegos de plataformas para implementar código de esta manera. El objetivo del juego consiste en encontrar los diversos trozos de una clave secreta (ocultos en el mobiliario de 32 habitaciones) que le permite al jugador irrumpir en la ciudadela del malvado científico Elvin. Los muebles están custodiados por robots y, ocasionalmente, por una inmensa bola negra. Algunos elementos del mobiliario contienen, asimismo, contraseñas especiales que, tras obtenerlas, le permiten al jugador volver a montar ascensores en las habitaciones y desconectar temporalmente los robots para poder pasar junto a ellos con toda seguridad. Estas contraseñas se entran a través de los terminales de ordenador situados en cada habitación. El jugador se desplaza de una habitación a otra mediante un ascensor y a través de una serie de pasillos. En la esquina inferior izquierda de la pantalla hay un mapa que muestra las habitaciones en las cuales se ha entrado hasta el momento y la posición actual del jugador.

Las contraseñas también se pueden obtener a partir de cuartos de codificación. Para recibir una contraseña, la habitación emite una cantidad de notas de altura variable, y el jugador debe acomodarlas por orden ascendente. Cuantas más contraseñas intente uno obtener, mayor será el número de notas que se toquen. En este sentido, *Impossible mission* es un juego de plataformas estándar. A diferencia de la mayor parte de los juegos de plataformas, el jugador no posee una cantidad de "vidas" establecida, sino que, en cambio, cada vez que el agente es "asesinado", incurre en una penalización de tiempo.

En este juego se le da un uso excelente al espacio de memoria ahorrado por los programadores. La característica más notable es la síntesis de voz. Los gráficos son, igualmente, muy buenos, y aunque las habitaciones no están repletas de ellos, los detalles individuales del agente, el mobiliario y los robots están dibujados con gran refinamiento. Todos estos elementos se conjugan para conformar un juego atractivo y muy bien trabajado.

No hay nada imposible
El objetivo de *Impossible mission* consiste en recorrer, mediante ascensores y pasillos, las diversas habitaciones de la guarida de un perverso científico. Tras entrar en las habitaciones, evitando a los robots, el jugador debe buscar entre los muebles partes de la contraseña que finalmente le permitirá entrar en el laboratorio del profesor Atombender. Ocasionalmente también podrá descubrir "somníferos", contraseñas que desconectan temporalmente a los robots

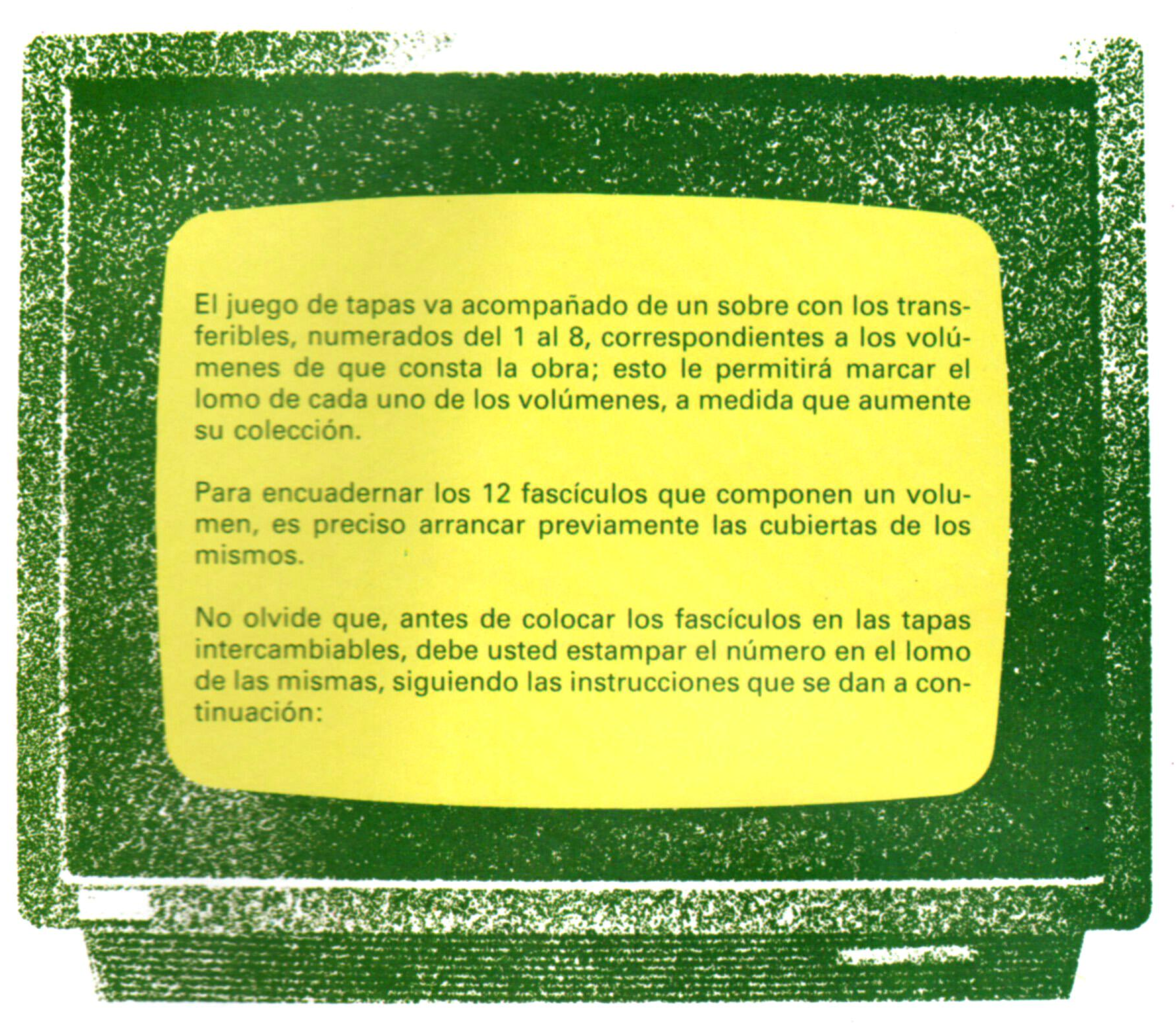
El ascensor desciende

Habitación de robots

Liz Heaney



Con el fascículo anterior se han puesto a la venta las tapas correspondientes al séptimo volumen.



El juego de tapas va acompañado de un sobre con los transferibles, numerados del 1 al 8, correspondientes a los volúmenes de que consta la obra; esto le permitirá marcar el lomo de cada uno de los volúmenes, a medida que aumente su colección.

Para encuadernar los 12 fascículos que componen un volumen, es preciso arrancar previamente las cubiertas de los mismos.

No olvide que, antes de colocar los fascículos en las tapas intercambiables, debe usted estampar el número en el lomo de las mismas, siguiendo las instrucciones que se dan a continuación:



1 Desprenda la hojita de protección y aplique el transferible en el lomo de la cubierta, haciendo coincidir los ángulos de referencia con los del recuadro del lomo.

2 Con un bolígrafo o un objeto de punta roma, repase varias veces el número, presionando como si quisiera borrarlo por completo.

3 Retire con cuidado y comprobará que el número ya está impreso en la cubierta. Cúbralo con la hojita de protección y repita la operación anterior con un objeto liso y redondeado, a fin de asegurar una perfecta y total adherencia.



Cada sobre de transferibles contiene una serie completa de números, del 1 al 8, para fijar a los lomos de los volúmenes. Ya que en cada volumen sólo aplicará el número correspondiente, puede utilizar los restantes para hacer una prueba preliminar.

9 788485 822836

10085